HZ BOOKS
华章教育

CENGAGE Learning

计 算 机 科 学 丛 书

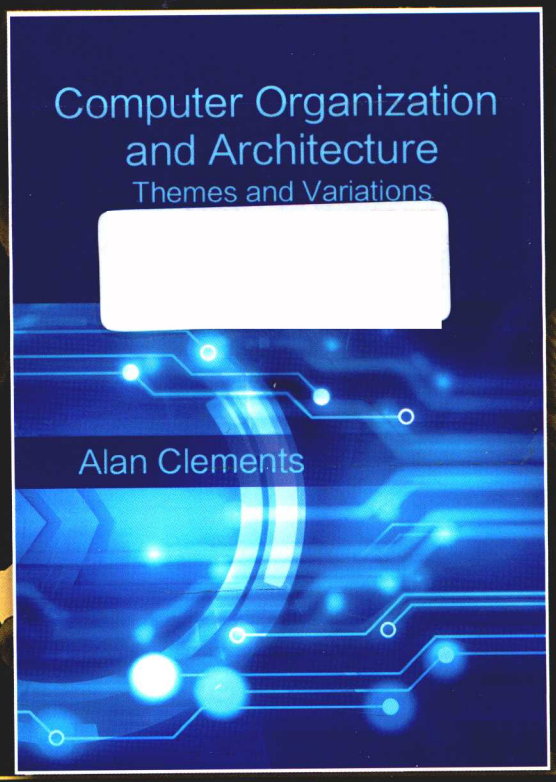
计算机存储与外设

[英] 艾伦·克莱门茨（Alan Clements）著

沈立 肖晓强 王苏峰 译

Computer Organization and Architecture

Themes and Variations



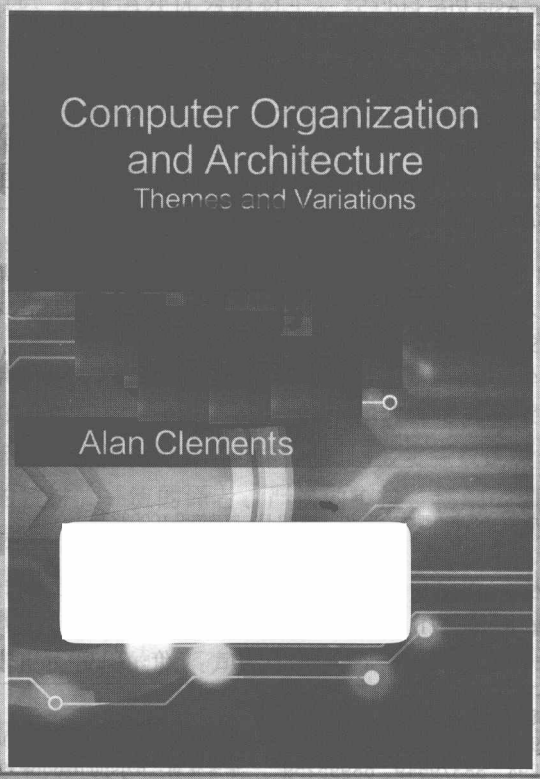
机械工业出版社
China Machine Press

计 算 机 科 学 丛 书

计算机存储与外设

[英] 艾伦·克莱门茨（Alan Clements）著
沈立 肖晓强 王苏峰 译

Computer Organization and Architecture
Themes and Variations



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

计算机存储与外设 /（英）艾伦·克莱门茨（Alan Clements）著；沈立等译．—北京：机械工业出版社，2017.1
（计算机科学丛书）
书名原文：Computer Organization and Architecture: Themes and Variations

ISBN 978-7-111-55748-7

I. 计… II. ①艾… ②沈… III. ①电子计算机－存储技术 ②电子计算机－外部设备 IV. ①TP333 ②TP334

中国版本图书馆 CIP 数据核字（2017）第 006320 号

本书版权登记号：图字：01-2014-0330

Alan Clements, Computer Organization and Architecture: Themes and Variations.
Copyright © 2014 Cengage Learning.
Original edition published by Cengage Learning. All Rights reserved.
China Machine Press is authorized by Cengage Learning to publish and distribute exclusively this simplified Chinese edition. This edition is authorized for sale in the People's Republic of China only (excluding Hong Kong, Macao SAR and Taiwan). Unauthorized export of this edition is a violation of the Copyright Act. No part of this publication may be reproduced or distributed by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.
Cengage Learning Asia Pte. Ltd.
151 Lorong Chuan, #02-08 New Tech Park, Singapore 556741
本书原版由圣智学习出版公司出版。版权所有，盗印必究。
本书中文简体字翻译版由圣智学习出版公司授权机械工业出版社独家出版发行。此版本仅限在中华人民共和国境内（不包括香港、澳门特别行政区及台湾地区）销售。未经授权的本书出口将被视为违反版权法的行为。未经出版者预先书面许可，不得以任何方式复制或发行本书的任何部分。
本书封面贴有 Cengage Learning 防伪标签，无标签者不得销售。

本书共 4 章，介绍了计算机系统中的存储器、总线、输入 / 输出等内容，具体包括 Cache 的组织、工作原理和虚拟存储技术，从静态半导体存储器到磁盘和光存储的各种存储技术，I/O 的基本工作原理和总线系统，以及一些支持多媒体系统的现代高速接口。

本书适合计算机科学、电子工程、电子与计算机工程及相关专业作为教学用书，也可供相关技术人员阅读参考。

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码 100037）
责任编辑：关　敏　　责任校对：董纪丽
印　　刷：三河市宏图印务有限公司　　版　　次：2017 年 3 月第 1 版第 1 次印刷
开　　本：185mm × 260mm　1/16　　印　　张：16.25
书　　号：ISBN 978-7-111-55748-7　　定　　价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

版权所有·侵权必究
封底无防伪标均为盗版
本书法律顾问：北京大成律师事务所　韩光 / 邹晓东

文艺复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum，Bjarne Stroustrup，Brian W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：（010）88379604
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

HZ BOOKS
华章教育

华章科技图书出版中心

译者序

Computer Organization and Architecture: Themes and Variations

计算机组成原理与系统结构是计算机科学与技术及相关专业的核心基础内容，其教学效果对于培养学生的计算机系统能力具有很大的影响。这两部分涉及的内容相互融合，密不可分。越来越多的国内外高校在教学设计、教学实施、教材编写时将这两部分内容结合在一起，并取得了明显的效果。

本书的英文版《Computer Organization and Architecture: Themes and Variations》不仅覆盖了单机系统的组成原理和系统结构的各个方面，还包括计算机的性能评价方法及多发射、粗粒度并行等内容。作者希望本书能够适合电子工程（EE）、电子与计算机工程（ECE）、计算机科学（CS）等不同专业的教学需要。书中围绕基本概念、指令集体系结构、处理器组成和能效、存储与外设，以及处理器级并行等五个核心问题将这些内容有条不紊地组织在一起，以便满足不同专业的教学需要。

综合考虑国内高校“计算机组成与结构”或类似课程的教学目标以及我们对本书的定位，中文纸质版分成了两本《计算机组成原理》和《计算机存储与外设》。

其中，《计算机组成原理》涵盖原书的前三部分（中文版《计算机组成原理》第 1 ～ 6 章分别对应原书第 1 章、第 2 章前 8 节、第 3 ～ 5 章和第 7 章）。

第一部分主要介绍计算机系统的组成、体系结构的基本概念。第 1 章介绍了计算机组成和结构的有关概念、计算机的发展历程，以及存储程序计算机，在读者面前呈现出基本存储程序计算机系统的形象。第 2 章则讨论了数据在计算机中的表示方法和运算方法。

第二部分介绍了指令系统的概念和实例。这部分包含三章内容。第 3 章首先介绍 ISA 的基本概念，之后以 ARM 指令集为例介绍了 ISA 设计时需要考虑的主要问题，如指令类型、寻址方式、数据表示等。第 4 章介绍了 MIPS——另一个经典的 RISC 指令集，以增加知识的深度和广度。第 5 章着重介绍了当前处理器为特定领域应用（比如多媒体应用）提供的支持，特别是指令集的支持。

第三部分介绍了处理器的实现以及一些影响处理器性能的因素。这部分只有一章，即第 6 章。它首先介绍了设计控制器的两种经典方法——微程序与组合逻辑，然后讨论了流水线技术、影响流水线性能的因素及一些可行的解决方法。

本书即为《计算机存储与外设》，涵盖原书的第四部分（中文版《计算机存储与外设》第 1 ～ 4 章分别对应原书的第 9 ～ 12 章），介绍了计算机系统中的存储器、总线、输入 / 输出等内容。第 1 章介绍了 Cache 的组织和工作原理，以及虚存技术。第 2 ～ 3 章涵盖了从静态半导体存储器到磁盘和光存储的各种存储技术。第 4 章介绍了 I/O 的基本工作原理以及总线系统，并描述了一些支持多媒体系统的现代高速接口。

中文纸质版没有收录原书中的门和数字逻辑、性能评价、多发射处理器、处理器级并行等内容（即原书 2.9 ～ 2.11 节和第 6、8、13 章），因为这些内容一般会在“数字逻辑”“计算机体系结构”“计算机系统性能评价”等课程中专门介绍。有兴趣的读者可以在中文版出版

社网站（http://www.hzbook.com）上找到相关章节的中文译文。

本书内容较多，翻译时间紧迫，尽管我们尽量做到认真仔细，但还是难免会出现错误和不尽如人意的地方。在此欢迎广大读者批评指正，我们也会及时在网上更新勘误表，便于大家阅读。

沈立

2016 年 12 月于长沙

前　言

Computer Organization and Architecture: Themes and Variations

21 世纪是科学和技术奇迹频出的时代。计算机已经做到了人们期望它做到的一切——甚至更多。生物工程解开了细胞的秘密，使科学家能够合成 10 年前无法想象的新药。纳米技术让人们有机会窥探微观世界，将计算机革命与原子工程结合在一起创造出的纳米机器人，也许有一天能够植入人体，修复人体内部的创伤。普适计算带来了手机、MP3 播放器和数码相机，使人们彼此之间能够通过 Internet 保持联系。计算机是几乎所有现代技术的核心。本书将阐述计算机是如何工作的。

从 20 世纪 50 年代起大学就开始教授这门被称为计算的学科了。一开始，大型机主导了计算，这个学科包括对计算机本身、控制计算机的操作系统、语言和它们的编译器、数据库以及商业计算等的研究。此后，计算的发展呈指数增长，到现在已包含多个不同的领域，任何一所大学都不可能完全覆盖这些领域。人们不得不将注意力集中在计算的基本要素上。这一学科的核心在于机器本身：计算机。当然，作为一个理论概念，计算可以脱离计算机而独立存在。实际上，在 20 世纪三四十年代计算机革命开始之前，人们已经进行了相当多的关于计算机的科学理论基础的研究工作。然而，计算在过去 40 年里的发展方式与微处理器的崛起紧密联系在一起。如果人们无法拥有价格非常便宜的计算机，Internet 也无法按照它已有的轨迹取得成功。

由于计算机本身对计算的发展及其发展方向产生了巨大影响，在计算的课程体系中包含一门有关计算机如何工作的课程是非常合理的。大学里计算机科学或计算机工程方向的培养方案中都会有这样一门课程。实际上，专业和课程的认证机构都将计算机体系结构作为一项核心要求。比如，计算机体系结构就是 IEEE 计算机协会和 ACM 联合发布的计算学科课程体系的中心内容。

介绍计算机具体体现与实现的课程有各种各样的名字。有人将它们叫作硬件课，有人管它们叫作计算机体系结构，还有人把它们叫作计算机组成（以及它们之间的各种组合）。本书用计算机体系结构表示这门研究计算机设计方法和运行方式的课程。当然，我会解释为什么这门课程有那么多不同的名字，并会指出可以用不同的方式来看待计算机。

与计算机科学的所有领域一样，计算机体系结构也随着指令集设计、指令级并行（ILP）、Cache 缓存技术、总线系统、猜测执行、多核计算等技术的发展而飞速进步。本书将讨论所有这些话题。

计算机体系结构是计算机科学的基石。例如，计算机性能在今天的重要性超过了以往任何时候，为了做出最佳选择，即便是那些购买个人电脑的用户也必须了解计算机系统的结构。

尽管绝大多数学生永远不会设计一台新的计算机，但今天的学生却需要比他们的前辈更全面地了解计算机。虽然学生们不必是合格的汇编语言程序员，但他们一定要理解总线、接口、Cache 和指令系统是如何决定计算机系统的性能的。

而且，理解计算机体系结构会使学生能够更好地学习计算机科学的其他领域。例如，指令系统的知识就能使学生更好地理解编译器的运行机制。

写作这本书的动机源于我在提赛德大学（University of Teesside）讲授计算机体系结构中级课程的经历。我没有按照传统方式授课，而是讲授了那些能够最好地体现计算机体系结构伟大思想的内容。在这门课程里，我讲授了一些强调计算机科学整体概念的主题，对学生的操作系统和C语言课程均有不小的帮助。这门课非常成功，特别是在激发学生的学习动力方面。

任何编写计算机体系结构教材的人必须知道这门课会在3个不同的系讲授：电子工程（EE），电子与计算机工程（ECE），计算机科学（CS）。这些系有自己的文化，也会从各自的角度看待计算机体系结构。电子工程系和电子与计算机工程系会关注电子学以及计算机的每个部件是如何工作的。面向这两个系的教材会将重点放在门、接口、信号和计算机组成上。而计算机科学系的学生大都没有足够的电子学知识背景，因此很难对那些强调电路设计的教材感兴趣。实际上，计算机科学系更强调底层的处理器体系结构与高层的计算机科学抽象之间的关系。

尽管要写出一本能够同时满足电子工程系、电子与计算机工程系和计算机科学系的教材几乎是不可能的，但本书进行了有效的折中，它为电子工程系和电子与计算机工程系提供了足够的门级和部件级的知识，而这些内容也没有高深到使计算机科学系的学生望而却步的程度。

本科计算机体系结构课可在三个不同层次上讲授：介绍性的、中级的和高级的。有些学校会讲授全部三个层次的内容，有些学校则将这些内容压缩为两个层次，还有一些学校只进行介绍。本书面向那些学习第一层次和第二层次计算机体系结构课的学生，以及那些希望了解微处理器体系结构当前进展的职业工程师。学习本书的唯一前提条件是读者应了解高级语言（如C）的基本原理和基本的代数知识。

由于本书覆盖了计算机体系结构的基础内容、核心知识以及高级主题，内容丰富，篇幅很大，适用于计算机体系结构相关的不同课程裁剪使用。综合考虑国内高校计算机组成与结构系列课程的教学目标和课程设置，中文版分成了两本《计算机组成原理》和《计算机存储与外设》。原书中关于门和数字逻辑、性能评价、多发射处理器、处理器级并行的内容，可在中文版出版社网站（http://www.hzbook.com）下载。——编辑注

本书特色

为什么还要编写一本计算机体系结构教材？计算机体系结构是一个很有吸引力的话题，它会介绍如何使用大量与非门那样的基本元件搭建一台计算机，也会介绍如何用常识来解决技术问题。例如，提升处理器速度的Cache在概念上并不比信封背面的记录复杂多少。同样地，所有处理器都使用了福特所发明的汽车制造技术：流水线或生产线。作者努力使本书的内容更加有趣或覆盖更多的主题，例如本书将介绍一些通过将氧原子从晶体的一端移到另一端来工作的存储设备。

用“它不是什么”来描述一个对象通常比用“它是什么”来描述更容易一些。本书并不关心微处理器系统设计、接口和外设的工程细节，当然也不会是一本汇编语言的入门教材。

本书的主题是微处理器体系结构而不是微处理器系统设计。就目前而言，计算机体系结构被定义为机器语言程序员所看到的计算机视图。这就是说，计算机体系结构并不关注计算机的实际硬件或实现，而仅关注它能做什么。我们也不会考虑微处理器的一些硬件和接口特征，除非它们对体系结构有明显的作用（例如 Cache、存储管理和总线）。

目标体系结构

任何体系结构教材的作者都会选定一个目标体系结构，作为讲授计算机设计和汇编语言程序设计基础知识的平台。教师们通常会热烈讨论究竟是用一款真正的商用处理器还是用一个假想的抽象处理器来讲授一门课。抽象处理器容易理解，学习曲线也比较浅，而且学生们通常会觉得理解一个真实处理器的细节知识很不值得。但另一个方面，实践工程都要适应真实世界中的各种约束。而且，一台真正的机器会告诉学生，工程师们为了制造出商用产品所必须做出的设计选择。

在 20 世纪七八十年代，DEC 公司的 PDP-11 小型计算机被广泛地用作教学平台。随着摩托罗拉 68K 等 16 位微处理器的出现，PDP-11 逐渐退出了课程教学。按照学术界的观点，68K（大致以早期的 PDP-11 为基础）是一台理想的机器，因为它的结构相对规整，学生也很容易用 68K 汇编语言编程。也许旁观者会希望使用绝大多数个人电脑都使用的、随处可见的 Intel IA32 系列处理器，让它在计算机体系结构教学中发挥重要作用，毕竟很多学生都有 Intel 处理器的亲身使用经验。但 80x86 系列处理器从未在学术界真正流行起来，因为每当一款新处理器发布，其复杂的结构都会以某种特定方式变化，这给学生带来了沉重的负担。一些教师采用高性能 RISC 处理器教学，比如 MIPS，这种处理器功能强大也容易理解。但这种高端 RISC 处理器多用于工作站，大多数学生对其不甚了解（老师们观察到，由于熟悉个人电脑，学生们通常更需要基于个人电脑的技术）。不过现在，RISC 处理器既用于高性能计算机，也用于绝大多数手机之中。

我选择 ARM 处理器作为介绍汇编语言和计算机组成的平台。这是一款性能高、结构优雅、易于学习的处理器。而且 ARM 处理器有很多开发工具，这意味着学生们可以写好 ARM 汇编语言源程序，并在实验室或家里的个人电脑上运行这些程序。

采用 Intel IA-64 结构的 Itanium 处理器是现代教材中目标体系结构的一个有力候选。这是一款功能极其强大但又极其复杂的处理器，不过它的基本结构却比 80x86 系列简单。Itanium 体系结构上大量创新性的特征验证了计算机体系结构课程中的许多概念——从数据栈到猜测执行，从流水线到指令级并行。因此，本书将在高性能计算部分[⊖]介绍这种处理器的一些特点。

本书并不是一部传统的计算机体系结构教材。它超出了传统课程的范畴，涵盖了许多有趣、重要且相关的内容。作者的一个重要目标是提供适合学生吸收的知识。很多时候，学生大学毕业后会发现他们的知识体系中有令人难堪的巨大空白。据我了解，目前还没有一本教材采用这种方法。例如，所有计算机体系结构教材都会介绍浮点运算，但却很少会讨论用于存储大量文本和视频信息的数据压缩的代码，更不会介绍 MP3 数据压缩这项工业界的核心

⊖ 这部分内容的电子版可在中文版出版社网站（http://www.hzbook.com）上下载。——编辑注

技术。类似地，计算机体系结构教材很少覆盖面向多媒体应用的体系结构支撑等内容。下面列出了本教材的一些特色内容。

历史

有关计算机体系结构的书籍通常会有一部分内容介绍计算机的发展历史。这些历史通常是不精确的，并会受到这一领域专家的批评。不过，我认为介绍历史的章节非常重要，因为有关计算机历史的知识会帮助学生理解计算机是如何发展的以及为何会这样发展。知道了计算机从哪里来，学生就能更好地理解它们在未来有可能怎样发展。在这本教材里，笔者给出了一段计算发展的简史，而与本书英文版配套的网络补充材料中则给出了更多的历史背景。

对操作系统的支持

操作系统与计算机体系结构密切地关联在一起。本书涵盖的体系结构内容（例如存储管理、上下文切换、保护机制等），即使是那些对操作系统进行研究的研究者，也会感兴趣。

对多媒体的支持

现代计算机体系结构背后最重要的驱动力是多媒体系统的发展及其对高性能和高带宽的无尽需求。本书介绍了如何面向多媒体应用优化现代体系结构。读者可以了解多媒体应用对计算机体系结构及对总线、计算机外设设计的影响，例如面向视听应用的硬盘。

输入 / 输出系统

今天的计算机不仅比它们的前身快得多，还提供了更多、更复杂的将信息输入计算机和从计算机中取出的手段。如果计算机仅与键盘、调制解调器和打印机连接，I/O 的重要性几乎可以忽略。现在的计算机经常与数字摄像机连接，需要传输大量数据。读者将会看到一些现代的高性能 I/O 系统，例如 USB 和 FireWire 接口，还会更深入地探究一些与输入 / 输出相关的内容，如握手机制和缓冲机制。

计算机存储系统

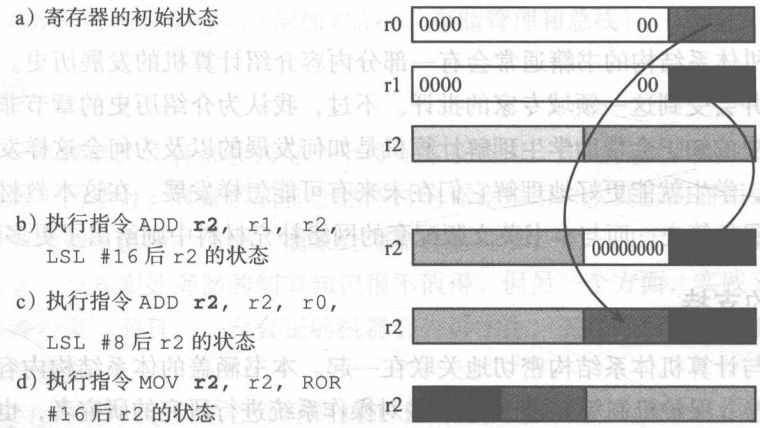
存储系统是计算机世界里的*灰姑娘*。没有高密度、高性能的存储系统，就不会有价格便宜的桌面系统，更不会有 32GB 存储容量的数码相机。本书将存储系统分为两章：前一章介绍半导体存储器，后一章介绍磁和光存储器。读者还会看到一些有趣的、正在兴起的存储技术，如相变存储器和铁电存储器。

方法

我之所以欣赏一本书，是因为它能够展现作者的风格与观点，希望本书也是如此。

笔者发现，插图和文字说明的质量是很多教材的一个不足之处。很多时候插图几乎没有注解，插图要表达的意思根本没有表现出来。本书的所有插图都由我自己完成，希望它们能够很好地阐释教材的内容。

本图描述了一个含有3条指令的代码段是如何修改同一个寄存器（r2）的内容的。这段代码的功能是从两个寄存器（r0和r1）中各取出一个字节，将它们拼在一起，放入第3个寄存器（r2）中。从图中使读者能够很容易看出数据是如何被处理的。



内容概要

本书即为中文版《计算机存储与外设》，分为4章，重点关注存储器、总线和输入/输出等内容。

第1章聚焦于两个相关的话题：Cache和虚拟存储。尽管存储系统的容量在过去几十年里增加得很快，但它们的速度或访问时间却并没有以CPU相同的速率改进。这种情况造成了一个瓶颈，即存储器无法以CPU处理数据的速度来提供数据。Cache技术已经发展到能用小容量快速存储器完成大容量快速存储器的工作，在困难的情况下已经尽量做到了最好。笔者会介绍Cache是如何工作的，并描述它们的原理特征。第1章还将介绍虚存，它将主存和磁盘存储器集成在一起。

第2章和第3章涵盖从静态半导体存储器到磁盘和光存储的各种存储技术。

第4章进行总结。其中首先介绍将信息送入计算机和从计算机中取出信息的技术，然后描述一些支持多媒体系统的现代高速接口。

总线是现代计算机的重要组成之一，它在系统的不同功能模块之间传递信息。实际上，总线是对计算机性能至关重要的一个部件。读者将看到计算机总线的结构、功能，以及多处理器系统中允许竞争总线的设备访问总线的方法。最后，笔者将介绍世界上最流行的高性能总线之一——个人计算机中的PCI总线。

补充材料和资源

本书为教师和学生提供了大量支持材料。这些补充材料都放在出版公司的网站上。要获得这些额外的课程材料，请访问www.cengage.com[⊖]。在cengage.com主页用页面顶部的搜索框查找本书，就可以找到访问这些资源的产品页。

⊖ 关于本书教辅资源，用书教师可向圣智学习出版公司北京代表处申请，电话：010-82862096/95/97，电子邮件：kai.yao@Cengage.com或asia.infochina@Cengage.com。——编辑注

教师资源

教师资源包括教师答案手册（ISM），包含教材中所有图表的完整 PowerPoint 幻灯片，以及一套所有公式和例题的幻灯片。

学生资源

学生资源包括：

- 一本详细介绍用 ARM 处理器汇编语言编程并在模拟器上运行这些程序的学生练习册；
- 附加学习材料，包括关于计算机历史的一章和卡诺图概述；
- 一些有用的链接，包括从 ARM 公司下载学生版 Kiel 模拟器的链接；
- 书中的代码；
- 本书作者网站（http://www.alanclements.org/）上发布的所有幻灯片的讲义。

致谢

没人能在真空中写出一部教材。所有学科都有其历史、背景和文化，计算机体系结构也不例外。一个作者要么随波逐流，要么沿着一个新的方向写作。没有以前出版的教材，例如当笔者还是学生时用来研究体系结构的教材，笔者也没有办法写出这本书。同样地，笔者也必须感谢数不清的对计算机体系结构知识体系做出了贡献的研究者。我的作用是收集所有这些内容并为学生学习构建一条知识路径。笔者必须决定哪些内容是重要的，哪些是可以忽略的，哪些趋势应该跟随，哪些趋势应被归结为背景知识，等等。不过，笔者还是要感激所有对知识体系做出了贡献的人。

许多人参加了这样一本复杂教材的出版工作，其中最突出的一位就是本书的策划编辑。他就是圣智学习（Cengage Learning）出版公司的 Swati Meherishi，是他开始了把最初的手稿转换为最终的优雅文字的漫长过程。策划编辑要有能力去了解书稿以外的内容，并知道本书如何才能适应复杂的市场。他对我有足够的信心，帮我度过了那段无休止写作和修改的时间。感谢 Swati 能容忍我。另一位本书创作的关键人物是项目开发编辑 Amy Hill。Amy 花费大量时间通读我的初稿，修改结构和内容组织方式。她给了我很好的建议和温和的指导。感谢圣智学习团队全体人员的努力工作和奉献，没有他们，这个项目还将停留为我硬盘上的一堆文件。还要感谢 Rose Kernan 和她 RPK Editorial Services 的团队对本书所有相关任务的高效管理，以及 Kristiina Paul 的精心研究和获得本书所有第三方材料的许可。

必须感谢所有帮助审阅和修订书稿的审稿人及文字编辑。他们提出了很好的改进建议，并且在我错误理解原始材料时为我指明了正确的方向。

感谢技术审稿人和那些仔细阅读本书并指出错误和疏漏的人。俄克拉何马州立大学的 Sohum Sohoni 审阅了全部书稿的技术内容。还要感谢南加州大学的 Shuo Qin，他完成了所有的习题并校正了教师答案手册中的错误。

许多教师在不同阶段对书稿进行了审阅。感谢他们提出的建设性意见。下面是要感谢的部分教师：

Mokhtar Aboelaze（约克大学）

Manoj Franklin（马里兰大学－帕克分校）

Israel Koren（马萨诸塞州立大学安姆赫斯特分校）

Mikko Lipasti（威斯康辛大学－麦迪逊分校）
Rabi Mahapatra（德州农工大学）
Xiannong Meng（巴克内尔大学）
Prabhat Mishra（佛罗里达大学）
William Mongan（德雷塞尔大学）
Vojin G. Oklobdzija（德州大学－达拉斯分校，Eric Jonsson 工程学院）
Soner Onder（密歇根技术大学）
Füsun Özgüner（俄亥俄州立大学）
Richard J. Povinelli（马凯特大学）
Norman Ramsey（塔夫斯大学）
Bill Reid（克莱姆森大学）
William H. Robinson（范德堡大学）
Carolyn J C Schauble（科罗拉多州立大学）
Aviral Shrivastava（亚利桑那州立大学）
Sohum Sohoni（俄克拉荷马州立大学）
Nozar Tabrizi（凯特林大学）
Dean Tullsen（加利福尼亚大学－圣迭戈分校）
Charles Weems（马萨诸塞州立大学）
Bilal Zafar（南加州大学）
Huiyang Zhou（北加州州立大学）

最后，感谢我的妻子 Sue 帮我修改书稿。尽管她没有技术背景，但她还是帮助我修改了文中一些英语使用有歧义和难懂的地方。

非常欢迎读者的反馈，无论是批评的还是赞扬的。对于那些能放到本书配套网站上，对学生学习有帮助的附加材料的建议，我会特别感兴趣。请将评论、关注和建议发到 globalengineering@cengage.com。也可以通过 alanclements@ntlworld.com 直接与我联系。

Alan Clements

穿越计算机体系结构之路

阅读一本书最简单的方法是从第一页开始读，直到最后一页。本书可作为“计算机存储与外设”课程的教材，以下的建议也许会帮助读者阅读本书。

尽管可以说本书中的所有内容都很重要，但它们不可能对每位读者都同等重要。一些读者也许已经熟悉了基础知识，可以略过本书的介绍部分。学生们也许还发现一些内容已经超出了课程考试范围，不过这并不意味着他们可以跳过这些内容。后面一些高级章节也许更贴近实践，或对学生今后的职业生涯更重要。我曾经问过学生：“如果雇主面试两个条件几乎完全相同的学生，其中一个超出课程范围读完了一些额外的资料，你们认为哪个学生更有可能获得这份工作？”

这里将给出一些如何阅读本书的建议，包括：

- 章节概要列出了本书的各个内容对不同课程的适用性；
- 面向初学者和高级读者的课程的内容组织；
- 本书与 IEEE CS/ACM 计算学科课程体系的关系；
- 本书中反复出现的主题。

章节概要

下表列出了本书的各个章节，指明了它们的内容是怎样符合教学计划的，以及这些章节中是否含有可作为初级或高级背景知识的内容。

章节	全部	重要	补充	初学者	高级读者
1 Cache 存储器和虚存	√			√	
2 主存储器	√				√
3 二级存储器	√				
3.4 安全存储和 RAID 系统	√				
3.5 固态盘		√	√		√
3.6 磁带			√		
3.7 光学存储技术		√	√		√
4 输入 / 输出	√				
4.1 I/O 基本原理	√			√	
4.5 总线	√			√	√
4.6.1 本地化仲裁和 VMEbus			√		√
4.6.2 分布式仲裁			√		√
4.7.1 PCI 总线			√		√
4.9 串行接口总线		√	√		

适合初学者和高级读者的方法

有人让笔者就阅读本书的方法给出一些建议。上面的内容对学生和教师使用本书均有帮助。笔者在这里给出两个建议，一个适用于初次学习这些内容的学生，另一个则适用于已经学习过数字逻辑或类似基础课程的学生。但这仅仅是建议。每门课程都不相同，每位老师也有自己的偏好。下面两种方案都不是细化的，因为并非所列出的每个章节都必须在课上讲授。

基础课程

第 1 章	全部
第 2 章	2.1
第 3 章	3.1　3.3.2　3.7
第 4 章	4.1　4.2　4.5

高级课程

第 1 章	全部
第 2 章	2.1　2.2　2.3　2.4
第 3 章	3.1　3.2　3.3　3.4　3.5　3.7
第 4 章	全部

与 IEEE CS/ACM 计算学科课程体系的关系

1991 年，ACM 和 IEEE 计算机学会发布了一份有关计算学科课程体系的报告（CC1991）作为计算机科学课程设置的指南。CC1991 为全世界所有大学提供了一份计算机科学课程设置的框架。这一工作是基于 ACM Curriculum'68 完成的。

计算领域没有一成不变的事物，ACM 和 IEEE 计算机学会在 1998 年开始将 CC1991 更新为 CC2001 以涵盖过去 10 年的变化。我曾在 Willis King 博士的带领下参加了 CC2001 标准体系结构部分的修订。

2007 年，为了反映当时的发展趋势，CC2001 开始进行内部修订。2007 年修订版中引入了一些新内容，但并没有包括那些全新的领域。在更新的学科课程体系标准中，我主要负责体系结构部分的草稿撰写工作，然后提交编委会审定。下面列出了计算机体系结构部分每个内容的学习目标。

中文版《计算机存储与外设》涵盖存储系统、总线和计算机外设的大部分内容。

数字逻辑和计算机算术 [核心]

学习目标：

- 用基本模块设计简单电路；
- 掌握二进制数的与、或、非和异或操作；
- 理解如何用二进制表示数字、文本、图像和声音，以及这些表示的局限性；
- 理解舍入效应是如何产生误差的，以及误差传播对链式计算精度的影响；
- 掌握如何压缩数据以减少对存储容量的需求，包括无损压缩和有损压缩的概念。

计算机体系结构 [核心]

学习目标：

- 介绍计算机从真空管到超大规模集成电路（VLSI）的发展；
- 理解指令集体系结构（ISA）的概念，从功能和资源（寄存器和存储器）使用等方面描述了机器指令的本质；
- 理解指令集体系结构、微体系结构、系统体系结构的关系，以及它们在计算机发展中的作用；
- 了解不同类型的指令——数据移动、算术运算、逻辑运算和流程控制；
- 掌握寄存器 – 存储器型指令集和装入 / 存储型指令集的区别；
- 掌握如何在机器级实现有条件操作；
- 理解子过程调用和返回的实现方法；
- 掌握系统在线可编程（ISP）的资源不足对高级语言和编译器设计的影响；
- 理解在汇编语言级如何将参数传递给子程序，如何创建和访问本地工作区。

接口和通信 [核心]

学习目标：

- 掌握开环通信和闭环通信的必要，以及使用缓冲来控制数据流的方法；
- 能够解释如何通过中断实现 I/O 控制和数据传输；
- 能够认识计算机系统中不同类型的总线，理解多个设备如何竞争并获得总线的使用权；
- 了解总线技术的发展，理解一些现代总线（包括串行和并行）的特点和性能。

存储系统组成与体系结构 [核心]

学习目标：

- 能够认识一台计算机中的存储技术，了解存储技术的变化方式；
- 掌握为诸如 DVD 等复杂数据存储机制制定存储标准的必要；
- 理解为何存储层次对于减少有效存储延迟是必要的；
- 掌握数据总线上传输的大多数数据都是填充到 Cache；
- 描述 Cache 的各种组织方式，掌握每种方法是如何在开销和性能之间进行折中的；
- 掌握多处理器系统中 Cache 一致性的必要性；
- 描述虚存的必要性，虚存与操作系统的关系，将物理地址转换为逻辑地址的方法。

功能性组织结构 [核心]

学习目标：

- 回顾如何使用寄存器传输语言描述计算机内部的操作；
- 理解 CPU 控制器是如何解释机器指令的——直接解释或将其解释为一段微程序；
- 掌握如何使用流水线通过指令重叠执行提高处理器性能；
- 理解处理器性能与系统性能之间的区别（即存储系统、总线和软件对总体性能的影响）；
- 掌握超标量体系结构是如何使用多个运算单元在每个时钟周期内执行多条指令的；
- 理解如何用 MIPS 或 SPECmarks 等指标衡量计算机的性能以及这些指标的局限；

- 掌握功耗与计算机性能之间的关系，以及将移动应用功耗降至最低的必要性。

多处理和其他体系结构 [核心]

学习目标：

- 讨论并行处理的概念以及并行性与性能之间的关系；
- 掌握用 64 位寄存器并行处理多个多媒体数据（例如 8 位 /16 位音频和视频数据）以提高性能；
- 理解如何在单个芯片上集成多个处理器以提高性能；
- 掌握将算法表示为适合在并行多处理器上执行的必要性；
- 理解专用图形处理器（GPU）是如何加速图形应用的性能的；
- 理解如何用电子设备配置和重新构建计算机的组成结构。

性能提升 [可选]

学习目标：

- 解释分支预测的概念以及用它提高流水线计算机性能的方法；
- 理解猜测执行是如何提高性能的；
- 给出超标量体系结构的详细描述，以及乱序执行时确保程序正确性的必要；
- 解释猜测执行机制，认识判断猜测正确性的条件；
- 讨论多线程技术的性能优势，以及那些使这种技术很难达到最大性能的限制因素；
- 掌握 VLIW 和 EPIC 体系结构的基本特点以及它们之间（以及它们与超标量处理器之间）的区别；
- 理解处理器是如何通过重新安排存储器 load 和 store 指令的顺序而提高性能的。

网络和分布式系统体系结构 [可选]

学习目标：

- 解释网络系统的基本组成，区分局域网（LAN）和广域网（WAN）；
- 讨论与分层网络协议设计有关的体系结构问题；
- 解释网络系统和分布式系统在体系结构上的不同；
- 掌握无线计算的特殊需求；
- 理解物理层和数据链路层角色上的不同，掌握数据链路层如何处理物理层的缺陷；
- 描述正在兴起的以网络为中心的计算领域，评价它们当前的性能、局限和近期的发展潜力；
- 理解网络层是如何检错和纠错的。

外设 [可选]

学习目标：

- 理解如何用数字形式描述压力等模拟量，以及使用有限状态表示是怎样导致量化误差的；
- 掌握多媒体标准的必要性，能够用非技术语言解释标准的要求；

- 理解为何多媒体信号通常需要通过无损或有损编码压缩来节约带宽；
- 讨论霍尔器件、压力计等传感器的设计、构造和操作原理；
- 掌握典型输入设备是如何工作的；
- 理解不同显示设备的工作原理和性能；
- 研究数码相机等基于高性能计算机的设备的工作过程。

新的计算方向 [可选]

学习目标：

- 掌握现代计算的底层物理基础；
- 理解物质的物理特性是如何制约计算机技术的；
- 掌握如何开发物质的量子特性以开发大规模并行性；
- 掌握如何用光完成特定类型的计算；
- 理解有机计算机是如何挖掘复杂分子的特性的；
- 了解存储设计的趋势，如相变存储器和铁磁存储器。

多次出现的主题

本书的一些内容会多次出现在计算机体系结构或其他课程中。它们一般是课程的基本内容，有时也会出现在计算机科学的其他领域。笔者力求完整地列出几个多次出现的主题以及它们在本教材中出现的章节。但这个列表无论如何也不能算作是完整的。

仲裁	4.6.1	4.6.2	4.9.2			
总线	4.5					
Cache 存储	1.1					
存储器	2.1					
协议	4.3	4.6.1				
状态图	4.8					
时序图	2.2.1	2.3	4.1	4.3	4.5.2	4.6.1	4.9.2
虚存	1.5					

作者简介

Computer Organization and Architecture: Themes and Variations

Alan Clements 出生于英格兰兰开夏郡，在苏克赛斯大学（University of Sussex）学习电子学。1976 年，当微处理器刚出现的时候，他在拉夫堡大学（Loughborough University）研究数字数据传输均衡器并获得博士学位。通过用微处理器解决均衡问题，他对计算机设计产生了兴趣并加入提赛德大学（University of Teesside）计算机科学系。

20 世纪 70 年代，有关微处理器设计实践的文献非常少，他出版了这一领域的第一本书。该书反响非常好，他又撰写了两本重要教材。《计算机硬件原理》(The Principles of Computer) 是一本本科生教材，全面地介绍了计算机硬件，其内容涵盖了从布尔代数到测量转速的外设等各个方面。为鼓励学生对计算机体系结构感兴趣，该书采用一种对学生友好的风格撰写。

20 世纪 80 年代，Alan 撰写了有关微处理器系统设计的权威教材，介绍了设计一款微处理器的全部阶段，并提供大量实际电路，弥合了学术与实践之间的巨大鸿沟。由于 Alan 在微处理器设计方面的贡献，1993 年摩托罗拉授予 Alan 提赛德大学终身教授。

多年以来，Alan 对计算机体系结构教学中的问题越来越感兴趣，越来越多地参与到计算机科学的教育活动中。2001 年，他担任了计算机学会国际学生竞赛主席（CSIDC），并于同年获得英国国家教学奖，这是英国高等教育的最高奖项。Alan 积极参加工程教育的前沿会议，并担任两本刊物的计算机科学教育专刊的客座编辑。

Alan 在 IEEE 计算机学会（CS）担任了多个职务，包括 CS 出版社主编，CS 第二副主席，教育活动委员会主席等。他还担任了伊拉克利翁和科罗拉多州立大学的客座教授。

Alan 积极参加学科课程体系设计，撰写了关于未来计算机体系结构教育的论文，参加了 CS/ACM 2001 计算课程体系项目。他为欧盟、英国政府、日立公司和希捷公司等提供咨询工作。

2007 年 Alan 获得 IEEE 计算机学会泰勒布斯（Taylor Booth）教育奖。

除了教学和写作之外，Alan 还对摄影感兴趣，他的作品曾数次公开展出。他还是一个私人飞行员，将他对飞行和摄影的爱好结合在一起。在 www.pbase.com/clements 上可以找到他的摄影作品。

2010 年 Alan Clements 从全职教学岗位退休，专心于写作和拍摄。

出版者的话
译者序
前言
本书导读
作者简介

第 1 章 Cache 存储器和虚拟存储器 …… 1
1.1 Cache 存储器概述 …… 4
1.1.1 Cache 存储器的结构 …… 6
1.2 Cache 存储器的性能 …… 8
1.3 Cache 的组织 …… 11
1.3.1 全相联映射 Cache …… 11
1.3.2 直接映射 Cache …… 15
1.3.3 组相联 Cache …… 19
1.3.4 伪相联、Victim、Annex 和 Trace Cache …… 23
1.4 Cache 设计中要考虑的因素 …… 25
1.4.1 物理 Cache 和逻辑 Cache …… 25
1.4.2 Cache 电气特性 …… 26
1.4.3 Cache 一致性 …… 26
1.4.4 块大小 …… 27
1.4.5 取指策略 …… 29
1.4.6 多级 Cache …… 30
1.4.7 指令和数据 Cache …… 32
1.4.8 写 Cache …… 33
1.5 虚拟存储器和存储器管理 …… 36
1.5.1 存储器管理 …… 36
1.5.2 虚拟存储器 …… 38
本章小结 …… 43
习题 …… 44

第 2 章 主存储器 …… 49
2.1 简介 …… 49
2.1.1 存储系统的原理和参数 …… 50
2.1.2 存储层次 …… 54
2.2 主存储器 …… 55
2.2.1 SRAM …… 55
2.2.2 交叉存储器 …… 63
2.3 DRAM …… 64
2.3.1 DRAM 时序 …… 68
2.3.2 DRAM 技术的发展 …… 71
2.4 只读存储器系列 …… 77
2.4.1 EPROM 系列 …… 78
2.5 新兴的非易失性技术 …… 84
2.5.1 铁电迟滞 …… 86
2.5.2 MRAM——磁阻随机访问存储器 …… 88
2.5.3 双向存储器 …… 89
本章小结 …… 91
习题 …… 92

第 3 章 二级存储器 …… 96
3.1 磁盘驱动器 …… 97
3.2 磁性和数据存储 …… 98
3.2.1 读 / 写头 …… 100
3.2.2 磁记录密度的极限 …… 101
3.2.3 磁盘数据记录原理 …… 102
3.3 磁盘上的数据组织 …… 109
3.3.1 磁道和扇区 …… 110
3.3.2 磁盘参数和性能 …… 113
3.3.3 SMART 技术 …… 118
3.4 安全存储和 RAID 系统 …… 120
3.5 固态盘 …… 126
3.6 磁带 …… 130
3.7 光学存储技术 …… 132
3.7.1 数字音频 …… 132
3.7.2 从 CD 中读取数据 …… 134
3.7.3 底层数据编码 …… 138

3.7.4 可记录光盘 …… 141
3.7.5 DVD …… 143
3.7.6 蓝光 …… 145
本章小结 …… 146
习题 …… 146
第 4 章 输入 / 输出 …… 149
4.1 I/O 的基本原理 …… 150
4.1.1 外围设备寄存器寻址机制 …… 154
4.1.2 外围设备访问和总线宽度 …… 155
4.2 数据传输 …… 158
4.2.1 开环数据传输 …… 158
4.2.2 闭环数据传输 …… 159
4.2.3 缓冲数据 …… 160
4.3 I/O 策略 …… 165
4.3.1 程序控制 I/O …… 166
4.3.2 中断驱动 I/O …… 167
4.3.3 直接存储器访问 …… 174
4.4 I/O 系统的性能 …… 176
4.5 总线 …… 177
4.5.1 总线结构和拓扑 …… 178
4.5.2 总线的结构 …… 179
4.6 总线仲裁 …… 185
4.6.1 本地化仲裁和 VMEbus …… 187
4.6.2 分布式仲裁 …… 192
4.7 PCI 和 PCIe 总线 …… 196
4.7.1 PCI 总线 …… 196
4.7.2 PCIe 总线 …… 203
4.7.3 CardBus、PC 卡和 ExpressCard …… 207
4.8 SCSI 和 SAS 接口 …… 210
4.9 串行接口总线 …… 215
4.9.1 以太网 …… 216
4.9.2 FireWire 1394 串行总线 …… 218
4.9.3 USB …… 225
本章小结 …… 232
习题 …… 232
参考文献 …… 237

Cache 存储器和虚拟存储器

任何足够先进的技术都与魔法无异。

——Arthur C. Clark

名字有什么关系？我们把玫瑰叫成别的名字，闻起来还是一样的芬芳。

——莎士比亚，《罗密欧与朱丽叶》

存储层次（Memory Hierarchy）

Cache 存储器（Cache Memory）使得计算机看似拥有了比实际更多的快速存储器。存储器应该具有非易失性（non-volatile）、廉价、快速、功耗低的特点。在现实世界中，每种存储技术都有其自身的特点，其中有些特点是相互矛盾的。例如，速度快的存储器往往昂贵，而速度慢的存储器往往便宜。存储系统使用了几种不同的技术，每种技术都发挥着不同的作用。总而言之，这些技术使整个存储系统表现为快速、非易失性、低成本。可以把各种存储技术分为不同的层，从而形成存储层次。

图 1-1 描述了典型的金字塔结构存储层次。在顶端的存储器速度最快（具有最短的访问时间），金字塔底部的存储器速度最慢。使用金字塔来表示存储层次的原因是，顶层的存储器的容量要小于底层的存储器容量。

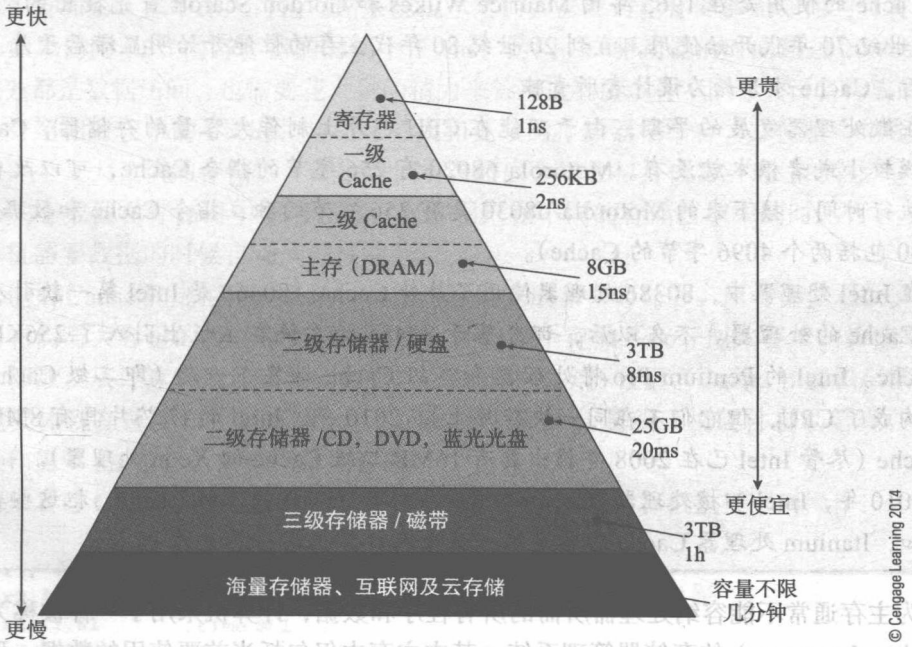


图 1-1　存储层次

片上的寄存器（register）是计算机中最快的存储器，它保存着处理器所需的工作数据。

寄存器的访问速度快，这使得 load/store 寄存器 – 寄存器型体系结构十分流行。寄存器是以与 CPU 相同的方法制备的，与 CPU 的时钟频率相同，且与 CPU 其他部分之间的数据通路都较短。此外，片上寄存器可以直接被 CPU 访问，而访问其他的外部存储器都需要一个过程，包括存储器管理、地址翻译以及复杂的数据缓冲和控制机制。因此，寄存器速度很快。但是，CPU 只有少量寄存器可以用来存储工作数据和状态信息。寄存器不能存放程序。

存储器层次结构中，在寄存器的下方就是 Cache，更准确地说是一级 Cache。Cache 的大小一般比主存小几个数量级，但真实程序的性质和数据的分布使得典型应用在 95% 以上的时间内只使用较小的指令和数据集合。历史上 Cache 曾经位于主板上，但芯片技术的进步使得人们可以在处理器芯片上实现大部分的 Cache。

图 1-1 显示了两级 Cache 存储器。如果被访问的数据不在一级 Cache 中，将访问下一级的存储层次，即二级 Cache。并不是所有的系统都有两级 Cache，有的系统还有三级 Cache。

如果数据不在 Cache 中，它必须从计算机的主存储器来获得，主存在图 1-1 中是 Cache 的下一级存储器。现代个人计算机和工作站大都使用 DRAM 来实现随机访问的主存储器。今天的个人计算机可以使用 1GB ～ 48GB 的 DRAM。

主存是易失性的存储器，因此程序和数据需要保存在非易失性存储器中。最便宜的存储机制之一是硬盘，它以磁信号的形式将数据保存在旋转的盘片上。一个硬盘可以存储超过 4TB 的数据。然而，硬盘的访问时间为 5ms 左右，虽然按照人的标准来说这已经很快了，但还是比主存慢 10^6 倍。今天，机械硬盘被更快、更可靠的固态硬盘（SSD）代替。这些技术将在第 3 章中讨论。

Cache 存储器的历史

Cache 的使用是在 1965 年由 Maurice Wilkes 和 Gordon Scarott 首先提出的。Cache 从 20 世纪 70 年代开始使用，直到 20 世纪 80 年代主存的性能开始明显滞后于片上寄存器以后，Cache 才开始为设计者所青睐。

在微处理器发展的早期，由于不能在 CPU 芯片上制作大容量的存储器，Cache 通常规模较小或者根本就没有。Motorola 68020 有 256 字节的指令 Cache，可以改善短循环的执行时间。接下来的 Motorola 68030 具有 256 字节的独立指令 Cache 和数据 Cache（68040 包括两个 4096 字节的 Cache）。

在 Intel 处理器中，80386 处理器使用了片外 Cache。80468 是 Intel 第一款引入 8KB 片上 Cache 的处理器。不久以后，部分基于 80486 的系统在主板上引入了 256KB 的二级 Cache。Intel 的 Pentium Pro 将处理器和二级 Cache 封装在一起（即二级 Cache 与处理器构成了 CPU，但它们不在同一块硅片上）。2010 年，Intel 的 i7 芯片具有 8MB 的片上 Cache（尽管 Intel 已在 2008 年推出具有 16MB 二级 Cache 的 Xeon 处理器）。

2010 年，Intel 四核处理器 Itanium 9300 系列有 24MB 的三级 Cache。把这些数据相加可知，Itanium 处理器 Cache 容量是第一代个人计算机内存总量的 24 倍。

因为主存通常不能容纳处理器所需的所有程序和数据，计算机采用了一种被称为虚拟存储器（virtual memory）的存储器管理系统，其中主存中仅包括当前要使用的数据，那些不使用的数据仍然保存在硬盘上。当处理器需要的数据不在主存中时，操作系统开始介入并使主存和磁盘之间交换一个页（page）的数据，其典型大小为 4KB ～ 64KB。虚拟存储器系统允

许用户运行比主存大得多的程序，且不会导致系统性能的显著下降。即由 512MB 的 DRAM 和 100GB 硬盘构成的虚拟存储系统的性能与具有 100GB 的 DRAM 的存储系统性能相当。虚拟存储器还提供了一种保护数据的手段。

存储层次的下一个层次是*光存储*，由 CD、DVD 或者蓝光光盘构成。光盘存储器将数据以凹痕的方式记录在塑料盘片上的螺旋轨迹中，并使用激光根据凹痕处是否有反光来读取数据。光盘的读取速度比硬盘的速度要慢，这主要是因为 CD 或 DVD 盘片的转速与硬盘片旋转速度相比差距太大。

图 1-1 中至少有一个数据是不准确的：光存储器的速度比磁记录存储器的速度慢，但其容量并不比后者大。CD 通常可以容纳 650MB 的数据，而蓝光光盘可以存储 25GB 的数据。几年前，该容量还相对较大，但硬盘技术的进步如此引人注目，在 10 年的时间中，硬盘的容量已经从 100MB 发展到超过 4TB（从 10^8B 增加为 4×10^{12}B，即容量增加了 4×10^4 倍。今天的硬盘能够存储比光盘更多的数据，在图 1-1 中光盘是因为速度慢而不是容量大才位于硬盘的下一个层次。

图 1-1 中光存储器的下面为*磁带*（magnetic tape）或*磁带存储器*（cartridge storage）。这些存储技术通过很长的磁带而不是旋转的磁盘来记录数据。磁带可以容纳大量的数据，但是它们的访问时间在几分钟甚至几小时的量级，所以它们只能作为后备存储器或者用于文档备份存储。

磁带的下一级是*互联网*（Internet）和*云存储*（cloud storage），它们可以提供远程分布式存储。数据存储在计算机外部，通常位于第三方提供的虚拟服务器中；即用户通过购买存储服务，通过互联网和 WWW 来存储信息。用户不需要知道数据到底存在哪里。重要的是，如何在灾难降临时保证数据的安全性。此外，只要连接到因特网，数据就可从任何地方访问。

在第 1 章和第 2 章中深入讨论存储系统技术之前，需要讨论存储器是如何与计算机系统关联的。如果访问 CPU 内寄存器的时间小于 1ns，访问 CD 驱动器的时间超过 200ms，虽然完成的任务都是数据访问，也需要花大量的精力来管理这种由多种不同存储设备构成的系统。

本章介绍相对较小的 Cache，与相对更大和更慢的存储器相结合，并使整个组合的系统看起来像一个又大又快的存储器。奇妙吧！本章然后介绍*虚拟存储器*对地址的管理，这使得用户不必关心数据在存储器中如何存储，允许用户在不同的存储区域中同时运行若干程序，且在计算机需要数据的时候自动从磁盘加载它。

本章包括 Cache 和虚拟存储系统，是因为它们完成相同的任务。两者都使用少量快速存储器，将它们与大容量的慢速存储器相结合，使得整个存储系统好似大容量的快速存储器。两种技术都涉及从计算机地址映射到数据在存储器中的实际位置。Cache 和虚拟存储器之间的区别主要体现在速度和控制机制上。Cache 操作需要 ns 量级，由硬件自动管理，而虚拟存储器操作需要 ms 量级，由操作系统管理。

下面来看看 Cache 和虚拟存储器在系统中的作用。假设计算机执行下列操作：

```
                        ;time = time + 1
LDR r1, = time          ;r1 指向“time”
LDR r2,[r1]             ;读 time
ADD r1,r1,#1            ;time 加 1
STR r2,[r1]             ;存 time
```

这些指令从内存读取操作数 `time`。CPU 所要做的就是将 `time` 的地址放到地址总线上，然后读取数据。原理上，该过程再简单不过了。图 1-2 展示了可以执行此代码的系统。计算

机给出的操作数地址（此例中为 time）是一个逻辑地址。将这个地址传给快速的 Cache，在那里试图访问数据。如果数据在 Cache 中，就从 Cache 中获取数据。否则，如果数据在较慢的主存储器中，该数据将从主存交给计算机和 Cache。该过程由硬件实现，并对用户和操作系统来说是不可见的。

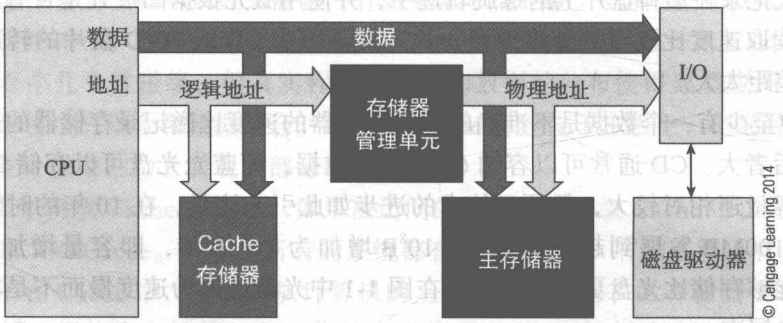


图 1-2 Cache 和虚拟存储器

有时，数据也不在主存储器中；它在硬盘上。当数据在磁盘上，虚拟存储器机制将其从磁盘拷贝到主存储器。它当然也被拷贝到了 Cache。从磁盘传输到主存储器的数据可以存放到主存储器的任意位置。这就需要存储器管理单元（memory management unit，MMU）将来自计算机的逻辑地址（logical address，即计算机认为数据所在的地方）转换为物理地址（physical address，即操作系统把该数据存放在存储器中的地址）。虚拟存储器管理是操作系统完成的主要任务之一，需要存储器管理单元的专用硬件与操作系统密切合作。

Cache 存储器的重要性

当今世界如何强调 Cache 的重要性都不过分。因为处理器速度和 DRAM 速度之间日益扩大的差距使得必须强制使用 Cache。

假设某 32 位高性能处理器具有超标量设计，在 1000MHz 的频率（即周期时间为 1ns）下每个时钟周期可以执行 4 条指令。为了能够在这个速度下工作，每个 ns 处理器都需要从 Cache 中获得 4×4=16 个字节的数据。如果数据不在 Cache 中，它需要通过 32 位总线从 DRAM 存储器中花费 50ns 获取，这样获取 4 条指令就需要 4×50ns=200ns。在该时间内，处理器可以执行 4×200=800 条指令。

下面将讨论图 1-1 中给出的存储层次的相关特性，然后介绍 Cache 是如何加快计算机操作的。本章的部分内容将介绍影响 Cache 性能的因素，以及如何通过优化 Cache 硬件和应用软件来提高系统速度。

本章的后半部分关注虚拟存储器以及如何从处理器给出的数据逻辑地址（或虚拟地址）映射到存储器中对应的数据物理地址。

1.1 Cache 存储器概述

在冯·诺依曼计算机中，存放程序和数据的主存储器应该尽可能快地满足 CPU 的要求。如果存储器不能在当前周期向 CPU 提供所需访问的数据或指令，存储器必须返回一个信号通知 CPU 等待。CPU 通过在机器周期中插入空闲（idle）或等待（wait）指令来阻止其执行

下一个操作。在等待状态，CPU 将停止正常操作，因此慢速存储器将严重降低其性能。在本节中，将介绍 Cache 如何显著提高处理器的性能，而不会带来太大的开销。图 1-3 再次展示了存储器的层次结构，并以时钟周期的形式给出了各层的延迟。

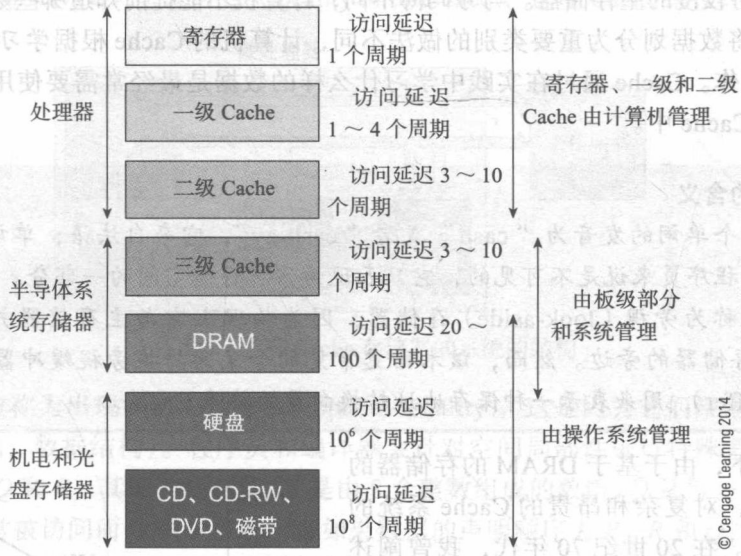


图 1-3 存储层次

20 世纪 90 年代早期，在各种期刊上常常可以看到这样的评论，“如果只是要 CPU 以更快的速度等待，则制造与使用高速 CPU 是没有意义的。”解决该困境的一个方案是使用更快的存储器以跟上 CPU 的速度。这听起来可能很容易做到，因为 CPU 和存储器的技术是相关的，它们都采用相同的设计和制造工艺。人们很容易认为一个具有 5ns 周期的 CPU 需要存储器的访存时间也达到 5ns。这种想当然的认知存在两个缺陷。首先，历史上处理器速度的增长率已远远超过了 DRAM 速度的增加，这在过去的 20 年中已经是事实。第二，CPU 的机器周期时间（即处理器执行一个读或写操作来访问外部存储器所需的时间）为 5ns，但其时钟周期可能只有 2.5ns，此时每个 CPU 的机器周期包括两个时钟周期。假设 CPU 在这 5ns 的机器周期时间中，将花费一个时钟周期来执行管理任务。处理器将需要在一个时钟周期内获得数据，该时间包括从数据地址可用到来自存储器的数据被锁定所需的时间。所以，要跟上 CPU 的存储器访问时间是 2.5ns。因此，这需要存储器的访问时间远低于其服务的处理器的机器周期时间。

尽管现代技术确实可以制造访问时间小于 5ns 的存储部件，但该设备的高昂成本使得其难以在大容量存储系统中应用。在 20 世纪 90 年代中期，个人计算机和工作站的大规模生产，为控制成本，要求使用已经测试过的主流存储部件（例如，64Mb DRAM 的访问时间为 50ns）。

Cache 并不神秘，它只是一个可由处理器快速访问的高速存储器。奇妙之处在于系统只拥有少量高速存储器（例如，某系统有 256KB 的 Cache 和 512MB 的 DRAM），并且希望处理器在 95% 的时间内都访问 Cache 而不是 DRAM。

Cache 可以通过日常生活中诸如日记、地址簿，或 iPhone 中的电话号码簿等类似物来理解。电话号码黄页中包含成千上万的电话号码，但没有人会真正带上一本。人们通常在号

码簿中只记录约 100 个电话号码。虽然这些号码可能少于电话号码清单总数的 0.0001%，但下一个电话来自号码簿的概率是很高的，就像人们经常会给朋友打电话一样。

Cache 的工作方式与上述电话号码簿的工作原理相同，它将经常访问的信息放入 Cache 而不是放入相对较慢的主存储器。与号码簿不同，计算机不能提前知道哪些数据是最可能需要的。与人们将数据划分为重要类别的做法不同，计算机的 Cache 根据学习原理（learning principle）来操作。Cache 通过在实践中学习什么样的数据是最经常需要使用的数据，然后将它们保存在 Cache 中。

Cache 名字的含义

Cache 这个单词的发音为“cash”或者“cash-ay”，它来自法语，单词的含义为隐藏。Cache 对程序员来说是不可见的，它只表现为系统存储空间的一部分。

Cache 也称为旁视（look-aside）存储器，因为物理上它与主存储器并行工作，看上去像在主存储器的旁边。然而，该术语更常见的含义为转换旁视缓冲器（translation look-aside buffer），用来表示一种保存地址转换向量的特殊 Cache。

理想情况下，由于基于 DRAM 的存储器的速度越来越快，对复杂和昂贵的 Cache 系统的需求应该消失。在 20 世纪 70 年代，我曾阐述了这个观点。时间证明我错了。图 1-4 显示了在过去 20 年中存储器和微处理器性能的发展趋势。存储器每年约增速 7%。处理器速度在这段时间内提升更快，每年增加 55% 左右的性能。由于处理器和存储器之间的性能差距已经扩大，Cache 存储系统今天的作用比其刚发明时的作用要重要得多。

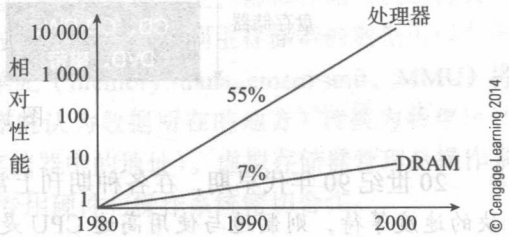


图 1-4 存储器性能发展趋势

Wulf 和 McKee 讨论了存储器和处理器性能变化具有不同速度带来的影响[⊖]。他们认为 DRAM 速度滞后于处理器的速度将最终被证明是限制计算机性能的条件。他们认为，处理时间是执行内部操作的时间加上访问外部存储器的时间的总和，因此，存储器访问时间部分将主宰处理时间。进一步提升处理性能将毫无意义。他们创造了“碰到存储墙（hitting the memory wall）”的概念，表明在传统微处理器系统设计中存在限制。

Cache 高速缓存本来没有什么内在价值。总线用来分布数据；硬盘用来存储大量的数据。Cache 的作用仅仅是隐藏了存储器延迟。如果存储器足够快，就可以不需要 Cache。

1.1.1 Cache 存储器的结构

Cache 存储器系统的一般结构如图 1-5 所示。Cache 存储体与处理器的地址总线和数据总线相连，并与更大容量的主存储器并行工作。Cache 中的数据在主存储器（即 DRAM）中也存在副本。

访问的局部性原理

回到前面的电话号码簿例子，向电话号码簿中添加一个朋友的号码并不会从电话号码黄页

⊖ William Wulf and Sally McKee, "Hitting the Memory Wall: Implications of the Obvious," *Computer Architecture News*, 23(1):20-24, March 1995.

中删除他的号码。考虑计算机访问某个存储器位置。各个给定的存储位置被访问的概率是不同的，因为有些地方比其他地方更容易被访问。由于程序和其数据结构的性质，处理器所需的数据往往在存储器中聚集在一起。例如，堆栈经常被访问，某些函数与其他函数相比经常被调用。这种现象被称为*访问的局部性*（locality of reference），这使 Cache 的使用成为可能。

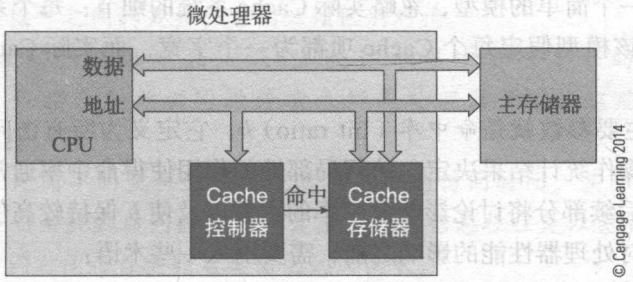


图 1-5　具有 Cache 存储器的系统的结构

一些地址被称为出现了*空间局部性*（spatial locality），这是因为它们聚集在相同的存储器区域内（例如，数据结构）。程序员和编译器会针对空间局部性进行特殊考虑。假设某程序包括变量 P、Q 和 R，其中 P 是整数，R 是由 8 个整数组成的数组，Q 是另一个整数。另外，假设 P 和 R 经常被访问而 Q 很少被访问。如果数据的声明顺序为 P、R 和 Q，则这两个经常访问的变量在存储器中是相邻的，可以被缓存在一起。

一些地址被称为出现了*时间局部性*（temporal locality），这是因为它们在短时间内被多次访问（例如，在一个循环体内的位置）。某循环，例如：

```
(for i = 0; i < 127; i++){
      P = P + R[i];
  }
```

将在一段时间内有规律地访问相同的变量。

局部性原理仅仅是种指导，而不能成为定律。某些程序同时表现出时间局部性和空间局部性，而有些程序没有。若程序要访问一个非常大的矩阵且数据分布是随机的，此时可能不会表现出空间局部性。某按照地理位置列表的邮购消费者数据库可能具有较好的空间局部性，这是因为一些团体由需要频繁服务的用户组成。一个简单的具有时间局部性和空间局部性的例子是产生内积的例子：

$$\sum a_i b_i$$

计算机连续访问 a_0，a_1，a_2，a_3，…，就会表现出空间局部性，因为这些连续的元素在内存中是相邻的。元素 a_i 和 b_i 在空间中存放的位置可能相距很远，但它们从访问时间上看是相邻的，因为 a_i 和 b_i 几乎在相同的时间被访问。

Cache 使用 Cache 控制器（cache controller）来确定 CPU 要访问的操作数是否在 Cache 中，是否需要从主存储器中调取。当给 Cache 控制器一个地址后，控制器返回一个信号（hit 或 miss）确定 Cache 是否命中。命中（hit）表明数据当前位于 Cache 中，失效（miss）表示数据不在 Cache 中，需要从主存储器中调取。

现代高性能系统具有多个级别的 Cache：一级 Cache、二级 Cache 和三级 Cache。一级 Cache 是容量最小但速度最快的 Cache。如果数据不在一级 Cache 中，就在二级 Cache 中查找。如果数据也不在那儿，则在三级 Cache 中查找。多级 Cache 的性价比较好，因为它们在

不增加最快 Cache 的容量的情况下提供了更好的性能。后续章节中将讨论多级 Cache。

1.2 Cache 存储器的性能

在判断加入 Cache 是否划算之前，需要知道增加 Cache 对计算机性能会产生多大的影响。这里首先介绍一个简单的模型，忽略实际 Cache 系统的细节：每个系统的 Cache 都不完全一样。特别是，该模型假定每个 Cache 项都为一个字宽，而实际 Cache 存放的是一个块（包含若干个字）。

Cache 系统的主要参数就是命中率（hit ratio）h，它定义为所有访问操作在 Cache 中命中的概率，由系统操作统计结果决定。访问局部性的作用使得命中率通常较高，基本保持在 98% 左右。本章的后续部分将讨论影响命中率的因素以及使 h 保持较高值的方法。

在计算 Cache 对处理器性能的影响之前，需要引入一些术语：

主存储器访问时间 t_m

Cache 访问时间 t_c

命中率 h

失效率 m

加速比 S

加速比（speedup ratio）是没有 Cache 时存储系统的访问时间与具有 Cache 时系统访问时间的比值。对于 N 次访存，如果没有 Cache，则总的访问时间为 Nt_m。

对于 N 次访存，如果有 Cache，总访问时间是 $N(ht_c+mt_m)$。失效率 m，定义为 $m=1-h$，因为访问要么命中要么不命中。因此，具有 Cache 的系统的加速比为：

$$S=\frac{N\times t_m}{N(h\times t_c+(1-h)t_m)}=\frac{t_m}{h\times t_c+(1-h)t_m}$$

该表达式假定所有操作均为访存操作，这是不正确的，因为处理器也要完成内部操作。后面还会谈到这一点。如果不关心主存储器和 Cache 的绝对速度的话，可以引入一个参数，$k=t_c/t_m$，它定义了 Cache 相对主存储器的速度比例。由 h 和 k 定义的加速比为：

$$S=\frac{1}{h\times k+(1-h)}=\frac{1}{1-h(1-k)}$$

图 1-6 显示了当 k=0.2 时加速比 S 与命中率 h 的关系曲线。正如大家理解的那样，当 h=0 时的加速比为 1，因为所有访存操作都访问了主存储器。当 h=1 时，所有访存操作都在 Cache 中进行，因此加速比为 $1/k$。

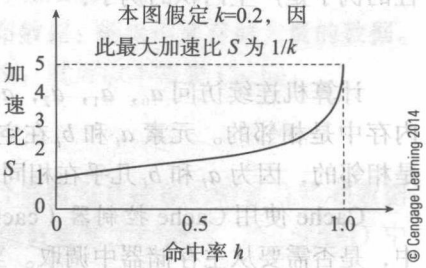


图 1-6 加速比与命中率

从图 1-6 中得出的一个最重要的结论是，加速比对命中率敏感。只有当 h 接近 90%，Cache 的性能影响才十分显著。该结论与常识是一致的。如果 h 低于 90%，访问主存储器的时间占较大比例，快速访问 Cache 带来的效果对系统性能的影响不大。

上述加速比 S 的公式适用于存储器和 Cache 可并行工作的系统，因为它假定每个存储周期开始时，同时（并行）访问主存储器和 Cache；也就是，访存地址同时交给 Cache 和主存储器。如果命中，终止对主存储器的访问。如果 Cache 不响应，数据将从主存储器中返回。

Cache 的复杂性

包含 Cache 的性能计算通常被简化，因为各种因素都会对真实系统产生影响。上面给出的 Cache 计算公式假定只从 Cache 或主存储器中获取一个字。实际上，当数据不在 Cache 中，存储系统通常会装载整个 Cache 块而不仅仅是其中的一个数据项。同时还必须考虑 Cache 读写操作之间的差异，它们必须区别对待。

此外，大多数高性能系统在一级 Cache 这个层次具有单独的数据 Cache 和指令 Cache，通过允许指令和数据同时传输来提高处理器带宽。这些 Cache 分别称为 D-Cache 和 I-Cache。

最后，这里假定简单的主存储器系统具有单一的访问时间。实际上，由 DRAM 组成的现代高性能存储系统的访问时间相当复杂，因为信息往往是以突发的方式访问，此时第一个存储器访问时间往往要比后续访问的时间长。

实际微处理器能够获得的加速比并不如上述公式得到的那样乐观。真实微处理器的操作速度由其时钟周期、每个存储器访问所需时钟周期数以及由于访问存储器而处于等待状态的时间确定。加快 Cache 的速度并不如减少等待时间重要。即使使用十分快速的 Cache，也不可能将访存时间减少到比不包含等待状态的一个总线周期还要短。

看看下面这个例子。

微处理器的时钟周期时间	10ns
每个总线周期所需最小时钟周期数	3
存储器访问时间	40ns
由于访存带来的等待时间	2 个时钟周期
Cache 访问时间	10ns
由于访问 Cache 带来的等待状态	0

这些数据表明，存储器访问需要（3 个时钟周期 + 2 个等待状态）× 10ns=50ns，访问 Cache 需要 3 × 10ns=30ns。该计算过程并不包含主存储器和 Cache 的实际访问时间。加速比为：

$$S = \frac{50}{30h + 50(1-h)} = \frac{50}{50 - 20h}$$

假设平均命中率为 95%，根据上式计算的加速比是 1.61（即 161%）。该结果比直接拿 Cache 和主存储器的访存时间来计算得到的结果（即 2.46）要小。

以上公式省略了不需访问存储器的内部操作的影响。下面看看德克萨斯仪器公司操作说明书中给出的既考虑非存储器访问操作又考虑访问 Cache 和主存储器中的数据时，微处理器的平均周期时间 t_{act}。

$$t_{act} = F_{int} \cdot t_{cyc} + F_{mem}[h \cdot t_{Cache} + (1-h)(t_{Cache} + t_{wait})]$$

式中：

F_{int} 为处理内部操作指令所占比例（以 70% 为例）；

F_{mem} 为访存操作指令所占比例（以 30% 为例）；

t_{cyc} 为处理器的时钟周期时间（以 10ns 为例）；

t_{wait} 为由于 Cache 失效导致的等待时间（以 50ns 为例）；

t_{Cache} 为 Cache 访问时间（以 5ns 为例）；

h 为命中率（以 0.9 为例）。

把给出的实际参数的例子代入公式计算，可得：

$$t_{act} = 70\% \times 10ns + 30\% \times [0.9 \times 5ns + 0.1(5ns + 50ns)] = 7ns + 3ns = 10.0ns$$

即使是这个结果也没有给出全部事实，因为实际系统不会一次仅在 Cache 和主存储器间移动一个字的数据。Cache 基本存储单元的容量不只是一个字，而是一块（line），它通常包括 4 ～ 64 个字节。当发生失效时，从存储器中会传输一块数据到 Cache。因此，失效开销不仅仅是调入一个数据，而是包括一块数据。

此时，也许可以对前面提到的存储墙进行讨论。假设某系统以周期为单位的指令平均执行时间 $Time_{avc}$ 为：

$$Time_{avc} = CPU_{use} \cdot t_{CPU} + Memory_{use}[h \cdot t_{Cache} + (1-h)(t_{memory})]$$

式中，CPU_{use} 表示非访存指令所占比例（以 80% 为例）；t_{CPU} 表示这类指令执行需要的时钟周期数（假定为 1）；$Memory_{use}$ 表示访存指令所占比例；t_{memory} 表示这类指令执行需要的时钟周期数（假定为 10）。如果命中率 h 为 0.95，则：

$Time_{ave} = 0.80 \cdot 1 + 0.20 \cdot 0.95 \cdot 1 + 0.20 \cdot (1-0.95) \cdot 10 = 0.80 + 0.19 + 0.10 = 1.09$ 个周期

经过一段时间，处理器的速度提高 10 倍，Cache 的速度提高 5 倍，DRAM 的速度提高 2 倍。CPU、Cache 与 DRAM 的访问时间之比不再是 1：1：10，而是 1：2：50。此时，

$Time_{ave} = 0.80 \cdot 1 + 0.20 \cdot 0.95 \cdot 2 + 0.20 \cdot (1-0.95) \cdot 50 = 0.80 + 0.38 + 0.50 = 1.68$ 个周期

假设第二种情况下，时钟周期时间是第一种情况的 1/10，加速比为 10.9/1.68=6.488。这表明时钟和 CPU 的速度快了 10 倍，而吞吐量只增加了 6 倍。

如果考虑 Cache 的写失效，可以重新审视 Cache 的性能。

考察 Cache 性能的其他途径

表示 Cache 性能有多种方法。有些人用失效率和失效开销来表示。有些人把 CPU 性能考虑进去。Cache 性能公式的差异取决于针对系统的假设。下面是几个 Cache 性能的计算公式：

1. 存储器暂停周期 = 访存指令百分比 × 失效率 × 失效开销对应的周期数
2. t_{CPU} = (CPU 执行周期 + 存储器暂停周期) × t_{cyc}
3. 平均访存时间 = AMAT = 命中时间 + 失效率 × 失效开销

实例

某计算机有独立的指令 Cache 和数据 Cache。数据 Cache 的命中率为 95%，指令 Cache 的命中率为 98%。失效开销（即访问主存储器需要的额外时钟周期数）为 100 个周期。处理器内在 CPI（即不考虑访存失效时的 CPI）为 1.5，25% 的指令为 load/store 指令。问：

- 平均失效开销为多少个时钟周期？

失效包括指令 Cache 和数据 Cache 的失效，即：

平均失效开销 = 0.02 × 100 + 0.25 × 0.05 × 100 = 2.0 + 1.25 = 3.25 个时钟周期

其中，0.02 和 0.05 分别是指令 Cache 和数据 Cache 的失效率，而 0.25 表示 25% 的指令需要访存。

- 整体 CPI 是多少？

整体 CPI 需要综合考虑 CPU 和存储器，即 1.5+3.25=4.75 个时钟周期。

● 如果：(a)Cache 是没有失效的理想 Cache，(b)Cache 不存在；加速比分别是多少？

(a) 具有理想 Cache 时，CPI 就是 CPU 对应的内在 CPI，即 1.5。加速比 = 有暂停的时间 / 没有暂停的时间 = 4.75/1.5 = 3.17。

(b) 如果不存在 Cache，每条指令的平均执行周期数为 1.5（指令执行时间）+100（获取指令的时间）+ 0.25 × 100(获取数据的时间)= 1.5 + 100 + 25 = 126.5 个时钟周期。此时，加速比为 4.75/126.5 = 0.038，即有 26 倍的性能下降[⊖]。

1.3 Cache 的组织

下面介绍 Cache 的组织和结构。一个 Cache 只是可用存储空间的一小部分。那么什么样的数据可以进入 Cache？数据会放在什么位置？相比于主存储器，Cache 存储系统的设计以及与计算机系统的集成都要更困难。部分是因为 Cache 速度快，部分是因为其复杂性高。事实上，直到 20 世纪 80 年代，只能在小型机和大型机上见到 Cache。直到 20 世纪 80 年代后期，随着 68020/30 以及 80386/486 微处理器的出现，Cache 才开始在个人计算机中出现。今天，已经可以将超过 10 亿个晶体管集成在芯片中，这样可以确保所有高性能处理器都包含一个大容量的片内 Cache。

Cache 设计的根本问题是如何构建一个存储体来存放来自大容量主存储器任意位置的一组数据元素。解决该映射问题的方法很多，但实际 Cache 系统通常使用组相联（set associative）方式进行组织。下面首先介绍两个 Cache 结构实例，以便读者理解组相联 Cache 的操作。

实例：加载 Cache

下一章将会介绍 DRAM。然而，对于本例来说，读者只需要知道从 DRAM 中读取一组数据时，第一个数据的访问时间要比后续数据的访问时间长。假设在主存储器中 DRAM 的时钟频率为 200MHz，它需要 6 个时钟周期来访问一组数据的第一个元素。后续每个元素将在一个时钟内被访问。存储总线为 64 位宽，Cache 每块需要存储 64 个字节；即 Cache 由 64 个字节的块组成。

在 Cache 失效后，加载一个 Cache 块需要多长时间？

1. 存储器为 64 位（8 字节）宽，读取 64 字节的 Cache 块需要 8 次存储访问。

2. 时钟频率为 200MHz，对应于 5ns 的时钟周期时间。DRAM 花费 6 个周期访问第一个 64 位的字，然后剩余的 7 个字每个均需要一个时钟周期。

3. 总时间为 6 × 5ns + 7 × 5ns = 30ns + 35ns = 65ns。

1.3.1 全相联映射 Cache

在设计任何一种存储系统时需要问的第一个问题是，基本数据单元的大小是多少？主存储器处理单位数据的大小与机器的基本字长相同。例如，具有 64 位寄存器的 64 位机器采用 64 位存储器。如果计算机需要少于一个字的数据，它首先读取整个字，然后再忽略它不想

⊖ 原书此例中出现多处错误，已修改。——译者注

要的位。

虽然计算机可以从 Cache 中读取一个字，但该字的大小并不是 Cache 基本存储单元的大小。Cache 基本存储单元为包含几个连续字的块（line）。假定 Cache 是按照字的粒度（granularity）来组织的。当访问某条指令时，如果它不在 Cache 中，则必须从主存储器中读取。然而，下一条指令又有可能会失效。因此需要比一个字大的 Cache 基本存储单元。

Cache 行话的噩梦

计算机科学使用的术语往往不一致；例如，典型的只读存储器（read only memory，ROM）同时也是随机访问存储器（random access memory，RAM）；再如，对于 68000 的程序员来说一个字是 16 位，而对于 ARM 编程者来说一个字为 32 位。

在 Cache 的世界中，使用不同的术语来描述给定地址对应的空间使得对 Cache 的理解变得一团糟。例如，Cache 存放数据的基本单元叫作块，有的英文文献叫作 line，有的称为 block。

一块数据由多个字（word）构成。一个块中的某个字的位置可以由其块地址（line address）以及块内偏移（block offset）给出。

在直接映射 Cache 中，块的地址有组地址（set address）和块地址（line address），它们是本书作者定义的术语，或者由标志（tag）和索引（index）来表示。下面的图例显示了这些术语的含义。

标志	索引	块偏移
组	块	字地址

© Cengage Learning 2014

Cache 块（cache line）由一系列连续的字构成，这样从 Cache 中可以连续读出几条指令从而不会发生失效。当发生失效时，包含该字的整个 Cache 块将从主存调入 Cache。Cache 块大小的最优值取决于 Cache 的总容量、代码的属性以及采用的数据结构。

人们希望 Cache 对存放在其中的数据不加限制；即 Cache 中的数据可以来自主存的任何地方。这种 Cache 使用相联存储器把数据存放在其中的任意位置，此时是根据其值（value）而不是根据其地址 / 位置（address/location）来访问数据。

图 1-7 说明了相联存储器的概念。每个数据项有两个值，关键字（key）以及数据元素；例如，最上面一行包含关键字 52B1 和数据 F0000。此时，数据没有排序，一个数据项可以保存在存储器的任意位置。访问关键字是检索数据的主要手段。访问相联存储器时需要将关键字输入存储器，然后将此关键字与存储器中所有关键字进行并行匹配。如果发现了该关键字，该数据就被检索到了（即与关键字关联的数据）。例如，计算机向图 1-7 中的系统输入关键字 F001。该关键字将同时交给存储器中的所有位置进行比较。由于给定 F001 会发生匹配（即命中），存储器将给出一个命中响应信号，同时将数值 42220 在数据端口输出。

相联 Cache 的工作原理看上去与图 1-7 中的机制相似，用关键字作为处理器访问的地址，数据就存放在该地址处。传统存储器与相联存储器的区别在于，传统存储器包含以 0,1,2，…编号连续存储的元素，而相联存储器包含的元素并没有排序或者按顺序存放。

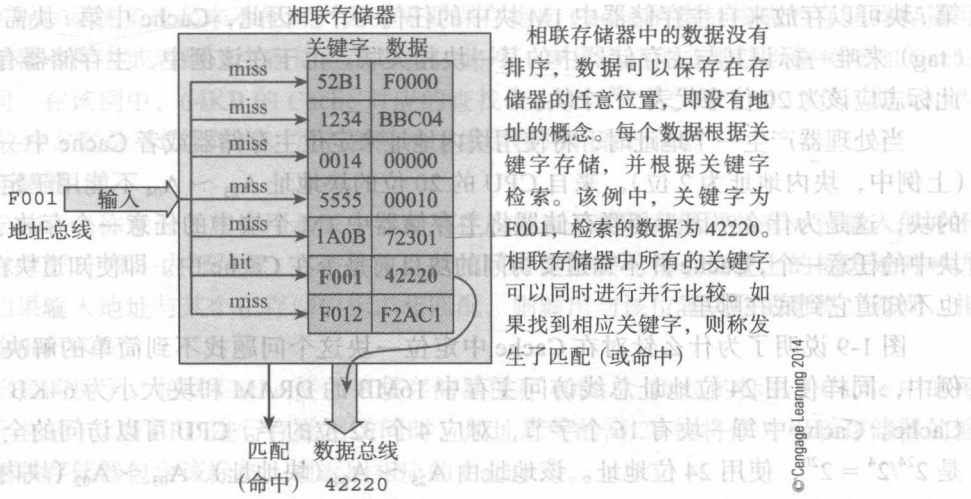


图 1-7 相联存储器的工作原理

下面详细讨论全相联 Cache 的部分细节。图 1-8 所示的相联存储器允许 Cache 中的任意一块都可以存放来自主存储器的任意一块数据。此例中，存储器被划分成包含两个字的块（每块两个字是为了简单起见，实际的 Cache 可能每块含有 8 个或更多的字）。

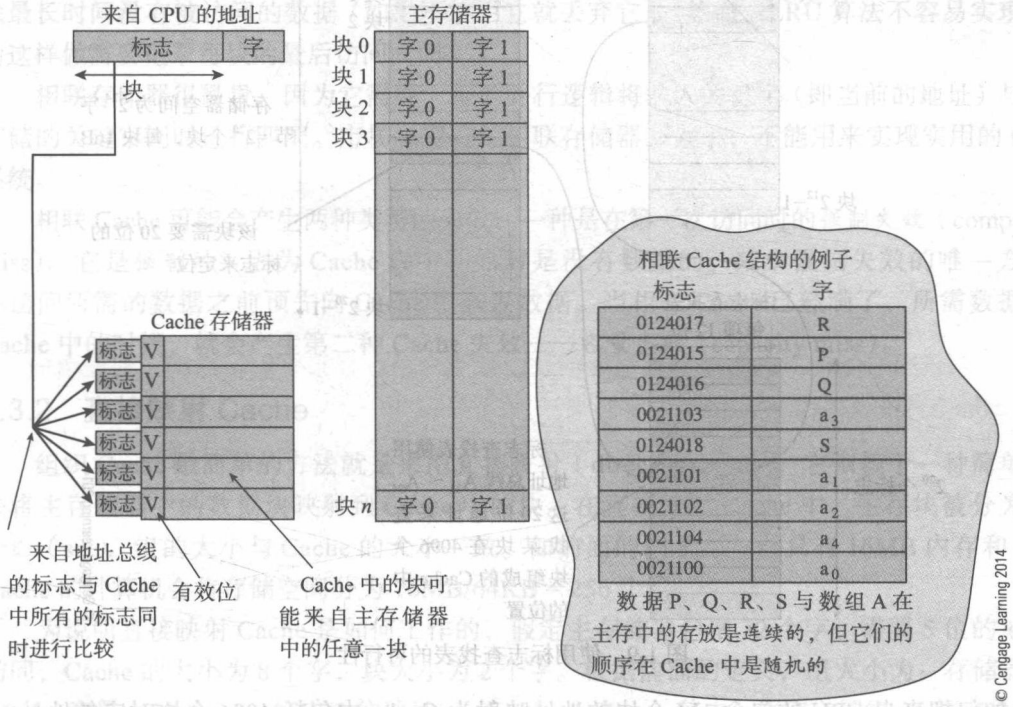


图 1-8 相联映射 Cache 的组成

相联 Cache 的大小可以任意，Cache 中块的数量与主存储器中块的数量之间没有关系。考虑主存储器为 16MB，相联映射存储器为 64KB。如果一块包含 4 个 32 位的字（即块大小为 16 字节），则主存储器由 $2^{24}/16 = 1M$ 个块构成，而 Cache 包含 $2^{16}/16 = 4096$ 个块。

相联 Cache 允许主存储器的任意一块数据存放在其任意一块中。上例中，相联 Cache 的

第 i 块可以存放来自主存储器中 1M 块中的任何一个。因此，Cache 中第 i 块需要一个标志（tag）来唯一标识其与主存储器中的某一块相关联。由于在该例中，主存储器有 1M 块，因此标志应该为 20 位来代表 2^{20} 个块。

当处理器产生一个地址时，将使用块内地址来定位主存储器或者 Cache 中一个字的位置（上例中，块内地址为 2 位）。来自 CPU 的 20 位的块地址 $A_{23} \sim A_{04}$ 不能用于定位 Cache 中的块，这是为什么？因为相联存储器将主存储器中 1M 个块中的任意一个存放于其 4096 个块中的任意一个，Cache 并不知道要访问的块目前是否在 Cache 中。即使知道块在 Cache 中，也不知道它到底在哪里。

图 1-9 说明了为什么针对在 Cache 中定位一块这个问题找不到简单的解决方案。在该例中，同样使用 24 位地址总线访问主存中 16MB 的 DRAM 和块大小为 64KB 的相联映射 Cache。Cache 中每一块有 16 个字节，对应 4 个 32 位的字。CPU 可以访问的全部块的总数是 $2^{24}/2^4 = 2^{20}$，使用 24 位地址。该地址由 $A_{23} \sim A_{04}$（块地址）、$A_{03} \sim A_{02}$（块内地址）以及 $A_{01} \sim A_{00}$（字节地址）组成。因此，需要 20 位的标志来唯一标识一块。

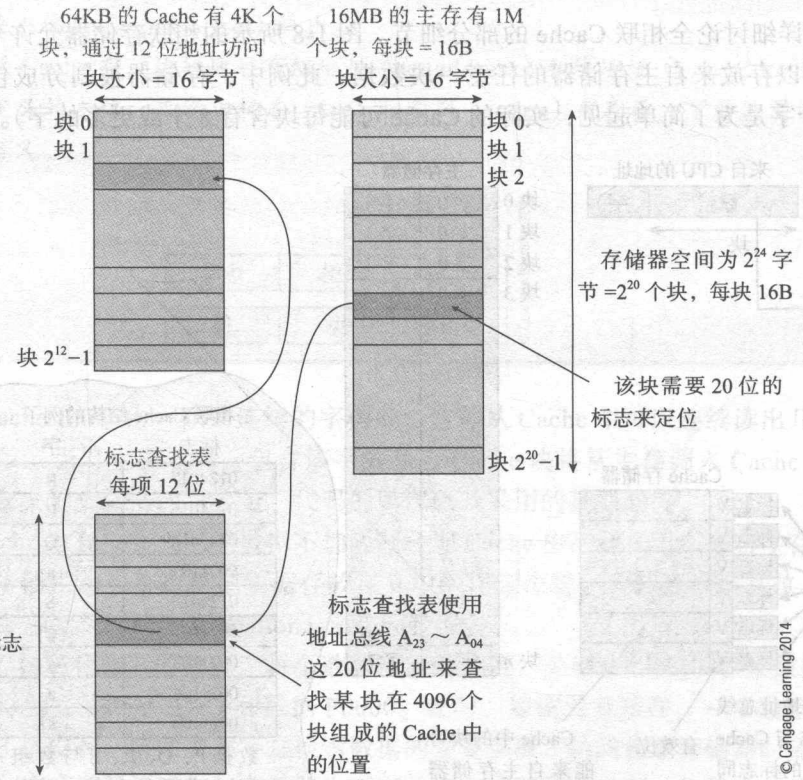


图 1-9 使用标志查找表的可行性

如何把来自 CPU 的包含 1M 个块的地址映射为 Cache 中包括 4096 个块对应的地址？图 1-9 给出了一种解决方案，该方案使用 2^{20} 个字、每个字 12 位的随机访问存储器来存储指向 Cache 块的 4096 个指针。该存储器是稀疏的，因为 1M 个存储位置中只有 Cache 中的 4096 块具有指针。其他位置没有指针是因为当前块在主存储器中而不是在 Cache 中。当然，为了实现这个机制，需要在查找表中每一行增加一个额外的位来表示当前块是否在 Cache 中（即“命中 / 失效”位）。

图 1-9 所示系统实现起来不便宜，因为需要保存指向 Cache 的指针构成的存储器容量可能与主存储器一样大或者超过主存储器的容量！此外，查找表必须非常快以避免增加 Cache 的访问时间。在该例中，64KB 的 Cache 对应的查找表有 1M 个字，每个字 12 位（即大小为 1.5MB），故该方案是不切实际的。可行的解决方案是使用相联存储器。

相联存储器

相联 Cache 名称来源于其使用相联存储器来存放标志。相联存储器具有 n 位输入但并不一定需要对应 2^n 个唯一内部位置。n 位输入交给相联存储器并与每个位置上的标志域同时进行比较。如果输入地址与某个正存储的标志相匹配，则输出与该位置关联的数据。否则，相联存储器输出不匹配。

继续前面的例子，相联 Cache 使用相联存储器保存 4096 个 20 位的标志（Cache 中的每块对应一个标志）。当 CPU 进行存储器访问时，地址总线的高 20 位将作为相联存储器的输入。如果相联存储器包含该标志，则返回相应的 Cache 块。

由于来自主存储器的块可位于相联 Cache 中的任意位置，当 Cache 满了会出现什么情况？需要删除哪个块来为新调入的块提供空间？实际上，相联 Cache 会保存每块上次被访问的时间，这样就可以将最久没有使用的块淘汰（或使用其他参数来决定将哪一块淘汰出 Cache）。Cache 的替换策略与虚拟存储器使用的类似（见下一节，例如，最近最少使用（LRU）、先入先出（FIFO），或随机策略等）。最近最少使用算法直观上最好，因为它旨在清除最长时间没有被访问的数据（如果不使用它就丢弃它）。然而，LRU 算法不容易实现，因为这样做需要记录每块的最后访问时间。

相联存储器很昂贵，因为它需要大量的并行逻辑将输入关键字（即当前的地址）与当前存储的关键字同时进行匹配。市场上现有的相联存储器都太小，不能用来实现实用的 Cache 系统。

相联 Cache 可能会产生两种类型的失效：一种是在第一次访问时的强制失效（compulsory miss）。它是强制的，因为 Cache 块中初始时是没有数据的。减少强制失效的唯一方法是在访问所需的数据之前预先向 Cache 中调入数据。当相联 Cache 已经满了，所需数据不在 Cache 中的时候，就会产生第二种 Cache 失效——容量失效（capacity miss）。

1.3.2 直接映射 Cache

组织 Cache 最简单的方法就是采用直接映射（direct mapping），它依赖于一种简单的算法将主存储器中的数据块映射到 Cache 中的块。在直接映射 Cache 中，主存块被分为若干个组（set），组的大小与 Cache 的大小一致。如前面的例子，一台具有 16MB 内存和 64KB Cache 的计算机会将存储空间分为 16MB/64KB = 256 个组。

为说明直接映射 Cache 是如何工作的，假定主存储器包括 32 个字，需要 5 位的地址来访问，Cache 的大小为 8 个字，块大小为 2 个字。根据前面的定义，组大小为：存储器容量/Cache 容量 = 32/8 = 4。给定 5 位地址 s_0、s_1、l_1、l_0 和 w，其中 s 位定义了组，l 位定义了块，w 位定义了字。图 1-10 展示处理器当前需要的字是如何根据组地址、块地址以及字地址进行访问的。为了方便讨论，这里只考虑组和块的访问，而不管一块中有多少个字。

图 1-10 所示的组织形式被称为直接映射 Cache，因为 Cache 中某块的位置与主存储器中块的位置之间有直接的关系。在图 1-10 给出的例子中，Cache 具有 2 位的块地址，因此具有 2^2=4 个块。如果直接映射 Cache 有 b 位块地址字段，则 Cache 就有 2^b 个数据块。

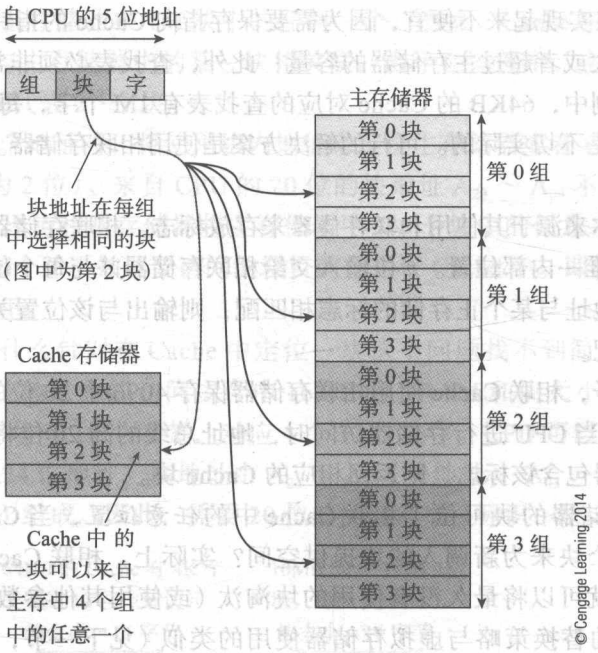


图 1-10 使用 5 位地址访问 Cache 和主存储器

当处理器产生一个地址，就会访问 Cache 中对应的块。例如，如果处理器生成 5 位地址 01100，则访问第 2 块。图 1-10 显示了主存储器中有 4 个块编号为第 2 块：第 0 组的第 2 块、第 1 组的第 2 块、第 2 组的第 2 块以及第 3 组的第 2 块。假设在这个例子中，处理器访问第 1 组中的第 2 块，显然的问题是：系统如何知道 Cache 中被访问的第 2 块是来自主存储器中第 1 组的第 2 块？

图 1-11 显示了直接映射 Cache 是如何解决这个问题的。在 Cache 中每块都有一个标志来标识该块来自主存储器中的哪个组。当处理器给出的访问地址中块地址为 3，Cache 中第 3 块对应的标志将发送给比较器。同时，处理器地址中的组字段也被发送给比较器。如果它们相同，则表明 Cache 中的块就是所需要的块，发生命中。如果它们不相同，则发生失效，对应 Cache 块中的数据必须更新。

当访问第 i 块并发生失效时，Cache 中原来的第 i 块要么被丢弃，要么被写回主存储器，这取决于主存储器的更新是如何组织实现的。后文将对 Cache 这方面的问题进行研究。

图 1-12 从另一种观点来描述直接映射 Cache，其中主存储器被当成一个组数 × 块数的矩阵，该例中为 4 块 ×4 组。矩阵的旁边是 Cache，它具有与主存储器相同的块数。当前 Cache 中的块与主存储器中阴影部分的块相对应。图 1-12 表明，Cache 中的块是如何从组号相同的主存储器的某组中获得的。

图 1-13 给出了直接映射 Cache 存储器系统的框架结构。Cache 存储体由保存数据的高速 RAM 构成。Cache 标志 RAM（cache tag RAM）是一种由高速随机访问存储器和数据比较器构成的特殊设备。Cache 标志存储器的地址输入是处理器访问的块地址（line address），该地址用来访问或指向包含该组标志的标志 RAM 中的位置。将该地址对应 Cache 标志 RAM 中的数据送往比较器并与地址总线上的组地址进行比较。如果处理器给出的组字段与被访问块的标志相匹配，Cache 标志 RAM 返回命中信号。

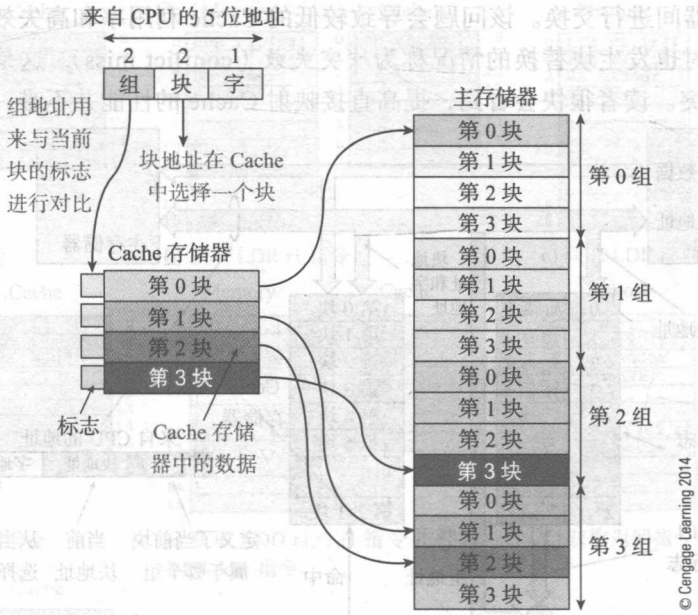


图 1-11 直接映射 Cache 的组成

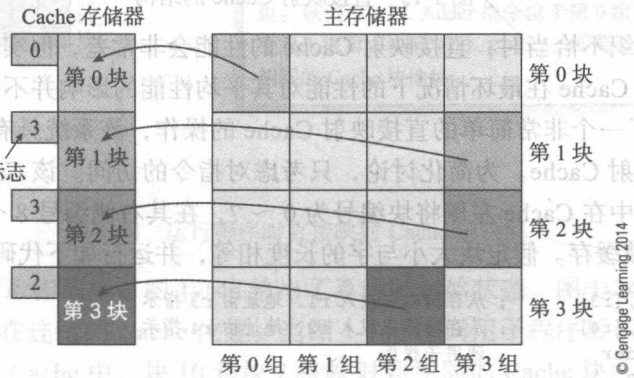


图 1-12 直接映射 Cache 的另一个视角

直接映射 Cache 不需要复杂的块替换算法。如果需要访问组 y 中的块 x 且发生了失效，主存储器组 y 中的块 x 将被加载到 Cache 中的第 x 块。当新块载入时，直接将失效位置对应的块淘汰出 Cache。

直接映射 Cache 的优点是其固有的并行性。由于保存数据的 Cache 存储器和 Cache 标志 RAM 是独立的，它们可以被同时访问。一旦从地址总线给出的标志字段与 Cache 标志 RAM 的标志字段相匹配，则命中，从 Cache 中获取的数据就是有效的。

直接映射 Cache 的缺点是它对被缓存数据的位置敏感。可以联系地址簿来理解，假设为每个字母保留 6 个位置。如果你已经有 6 个朋友的姓是以 S 开头的，那么下一次再碰到一个人的姓是以 S 开头时就会出现问题。这很烦人，因为给 Q 和 X 预留的位置可能完全是空的而不能用。由于只有一个编号为 x 的块能够放在 Cache 中，访问主存储器中其他不同的组中块号为 x 的块将导致 Cache 中第 x 块被替换。

即使 Cache 没有存满，在访问主存储器不同组中相同块号的两个块时，会导致块在

Cache 和主存储器间进行交换。该问题会导致较低的 Cache 利用率和高失效率。这种在即使 Cache 没有存满时也发生块替换的情况称为冲突失效（conflict miss），这是由于新块和原缓存块之间存在冲突。读者很快会看到，提高直接映射 Cache 的性能并不难。

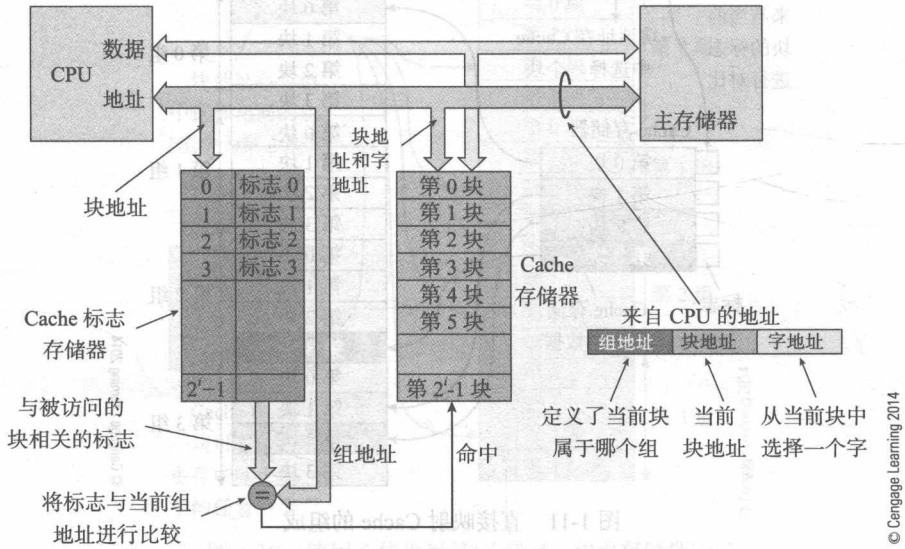


图 1-13 直接映射 Cache 的结构

虽然在数据组织不恰当时，直接映射 Cache 的性能会非常差，但实际程序的统计测量结果表明，直接映射 Cache 在最坏情况下的性能对其平均性能的影响并不明显。

图 1-14 展示了一个非常简单的直接映射 Cache 的操作，该系统具有 16 个字的主存储器和 8 个字的直接映射 Cache。为简化讨论，只考虑对指令的访问。该 Cache 可以保存来自两个组的一个块。图中在 Cache 左侧将块编号为 0 ～ 7，在其右侧编号 8 ～ 15，用来表明主存储器中 8 ～ 15 块被缓存。假定块大小与字的长度相等，并运行如下代码：

```
        LDR  r1,[r3]      ; 从存储器中载入 r1，地址由 r3 指示
        LDR  r2,[r4]      ; 从存储器中载入 r2，地址由 r4 指示
        BL   Adder        ; 调用子程序
        B    XYZ          ;
Adder   ADD  r1,r2,r1     ; r1 加 r2 结果放入 r1
        MOV  pc,lr        ; 返回
```

奇怪但是真的：直接映射 Cache 的性能可能优于相联 Cache

假设有一个很小的直接映射 Cache，只能存储 4 个块。假定某循环要访问 PQRST（5 次访问）。访问存储器的顺序将是 PQRSTPQRSTPQRSTPQ……

在 Cache 中不能保存整个序列。P、Q、R 和 S 被缓存后，T 替换 P，Cache 内容变成 T、Q、R 和 S。在下一步，P 取代 T 后，Cache 内容又变回了 P、Q、R 和 S。一轮循环下来，命中（h）和失效（m）的序列为 h、h、h、m 和 m。

如果该 Cache 为采用 LRU 算法的全相联 Cache，则序列为 P、Q、R 和 S。下一轮将替换 Cache 中最古老的元素 P 得到 T、Q、R 和 S。下一轮访问的元素为 P，使用 LRU 算法，P 将取代 Q。这样，每个新元素都会导致一次失效，全相联 Cache 会发生没完没了的失效。

Cache 块的来源有两个：每组中一个。左边的数字，0,1,2,…,7 对应于第 0 组，右边的数字 8,9,…,15 对应于第 1 组。该例中，BL Adder 指令位于第 0 组的第 2 块，然后被位于第 1 组第 2 块的 ADD r1,r2,r1 指令替换，因为它们具有相同的 Cache 块地址。

图 1-14　运行程序时直接映射 Cache 的快照

图 1-14 只显示了取指周期。图 1-14a 给出了系统的初始状态。图 1-14b ～ d 显示取前 3 条指令，每条都存放在连续的 Cache 位置。当图 1-14d 中调用子程序后，转向地址为 10 的指令。在该直接映射 Cache 中，块 10 和块 2 将映射到相同的 Cache 块。之后，在图 1-14e 中，ADD 覆盖了 Cache 块 2 中的 BL 指令此时发生了冲突失效，由于目的位置已经被占用，数据不能调入 Cache，除非该位置被空出来。

图 1-14f 中，第 11 块中的 MOV pc,lr 指令被装入 Cache 中第 3 块。最后，在图 1-14g 中，程序返回，第 3 块中的指令 B XYZ 被装入 Cache 中的第 3 块，替换掉以前的块。

图 1-14 表明，即使在一个简单的系统中，直接映射 Cache 中的元素可以很容易地被替换。如果上述这段代码循环执行，频繁替换 Cache 中的数据将降低性能。

1.3.3　组相联 Cache

上一节介绍的直接映射 Cache 很容易实现，不需要块替换算法。然而，它不允许来自不同组但具有相同块号的块被同时缓存。全相联 Cache 对块存放的位置没有限制，但它需要考虑一旦 Cache 已满将替换哪个块。此外，容量大的全相联 Cache 实现起来十分昂贵。组相联（set-associative）Cache 结合了上述两种 Cache 的优点，实现起来代价也不高。因此，它是计算机上常见的 Cache 组织形式。

实例：Cache 的容量

四路组相联 Cache 使用 64 位的字。每个 Cache 块包括 4 个字。1 组有 8192 个块。Cache 的容量有多大？

1. Cache 每块有 4 个 64 位的字，故每块为 32 字节。

2. 有 8192 个块，意味着直接映射 Cache 有 $8192 \times 32 = 2^{18}$ 字节，即 256KB。

3. 相联度为 4 意味着有 4 路直接映射 Cache 并行工作，因此 Cache 总容量为 $4 \times 256KB = 1MB$。

直接映射 Cache 只为每个块 *i* 分配了一个位置。如果使用两个直接映射 Cache 并行工作，就可以使块 *i* 进入它们中的任意一个。如果使用 *n* 个直接映射 Cache 并行工作，就可以使块 *i* 进入 *n* 个位置中的任意一个。这就是 *n* 路组相联 Cache。

在 *n* 路组相联 Cache 中，给定块可载入 Cache 中 *n* 个可能的位置。通常情况下，*n* 的范围是 2 ～ 8。图 1-15 展示了一个四路组相联 Cache 的结构，它由 4 个并行操作的直接映射 Cache 构成。在此情况下，块 *i* 可以载入 4 个直接映射 Cache 中的任意一个。因此，具有相同块号的不同块在调入 Cache 时产生的冲突将大大减少。这种方式称为相联（associative），因为来自处理器的块地址被并行送给每个直接映射 Cache。然而，一般并不会对成千上万的存储单元同时进行搜索，而只对 2 ～ 8 个直接映射 Cache 进行并行访问。每个 Cache 的比较结果（即命中）将被交给或门，其输出结果表示是否有一个 Cache 命中。

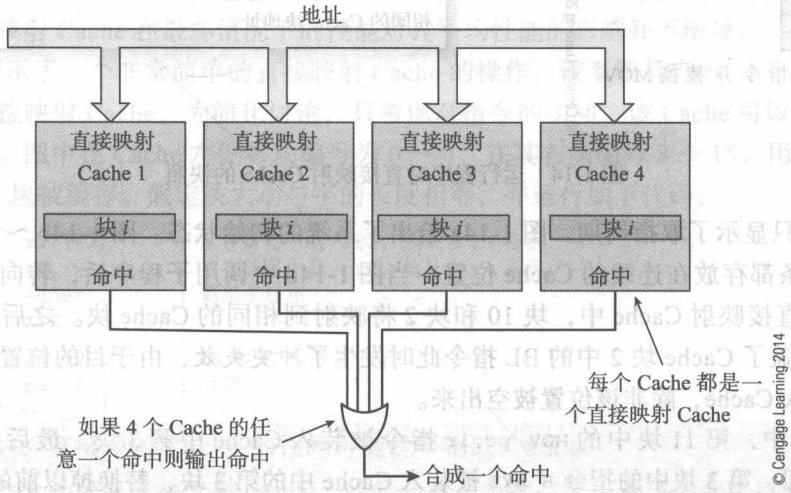


图 1-15 组相联 Cache 的组成

图 1-16 使用组相联 Cache 重复了图 1-14 中给出的例子。两个例子基本相同，只是组相联 Cache 中的每个直接映射 Cache 只有 4 块，但总容量还是 8 块。图 1-16 中所示为一个二路组相联 Cache，块可以保存在上面或者下面的直接映射 Cache 中的任意一个。

图 1-16 中前几步是一样的，直到图 1-16e，当地址为 10 的 `ADD r1,r2,r1` 指令映射到第 2 块（组大小为 4）时，该块已经被 `BL Adder` 指令占用。在相联的第 2 个 Cache 的相应位置是空闲的，因此该指令可以缓存在下面的 Cache 的第 2 块，而不必替换上面 Cache 中的块。图 1-16f，`MOV pc,lr` 指令的块号为 3，保存在上面的 Cache 中。然而，当执行主存储器第 3 块中的 `B XYZ` 指令时，上面的 Cache 的第 3 块已被占用，故它被缓存在下面 Cache 的第 3 块中。

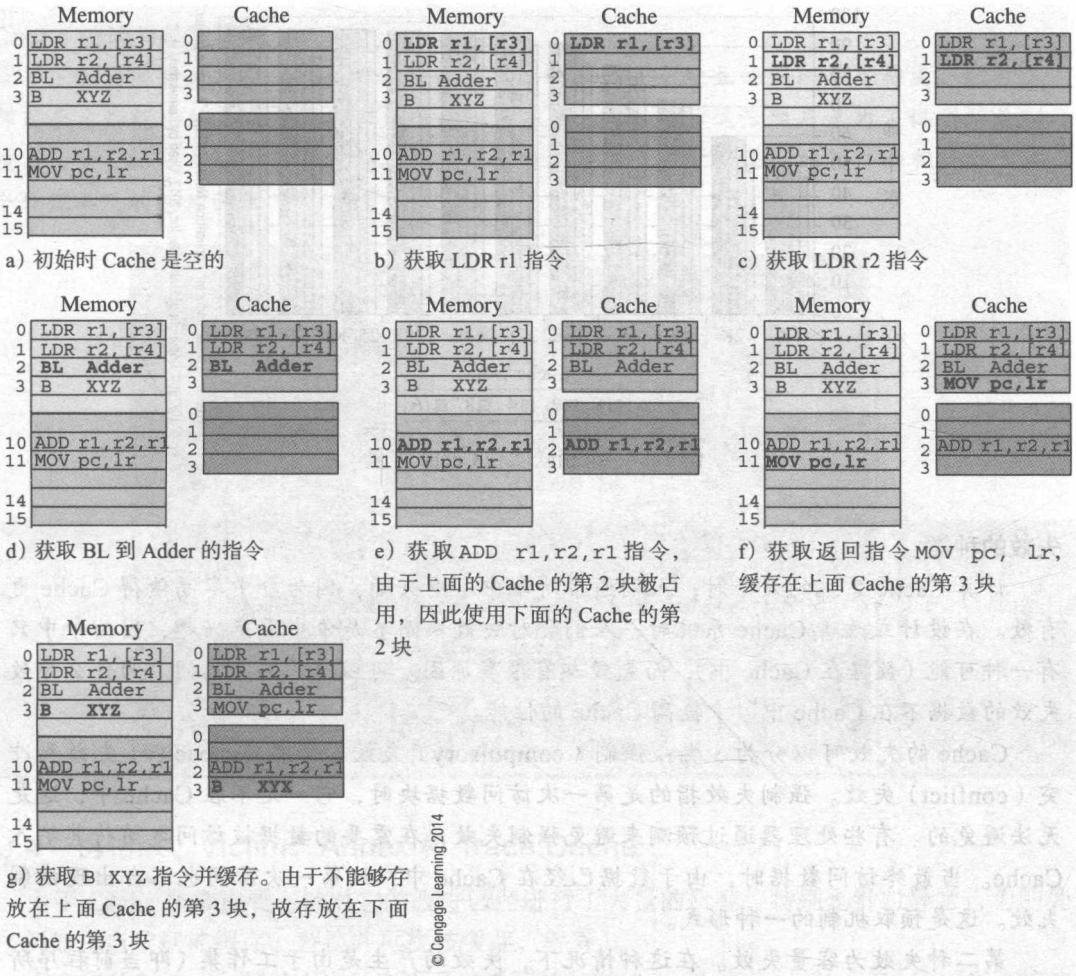


图 1-16 组相联 Cache 的行为

来自 IDT 应用笔记 AN-07《Cache 标志 RAM 芯片简化 Cache 存储器设计》的表 1-1 展示了 Cache 的组织形式对失效率的影响。该失效率已经与直接映射 Cache 的失效率相除进行了归一化，用来表示相对于直接映射 Cache 的结果。四路组相联 Cache 的结果要比直接映射 Cache 的结果好 30%。进一步增加相联度对于性能的改善有限。

表 1-1 Cache 的组成对失效率的影响

Cache 组成	归一化的失效率
直接映射	1.0
二路组相联	0.78
四路组相联	0.70
八路组相联	0.67
全相联	0.66

来自 Freescale 半导体公司应用笔记的图 1-17 展示了不同容量的 Cache、使用 GCC 编译器时相联度和命中率之间的关系。可以看出，Cache 容量比较小时，相联度是一个重要因子。当 Cache 容量达到 256KB 时，相联度的作用就不十分显著了。

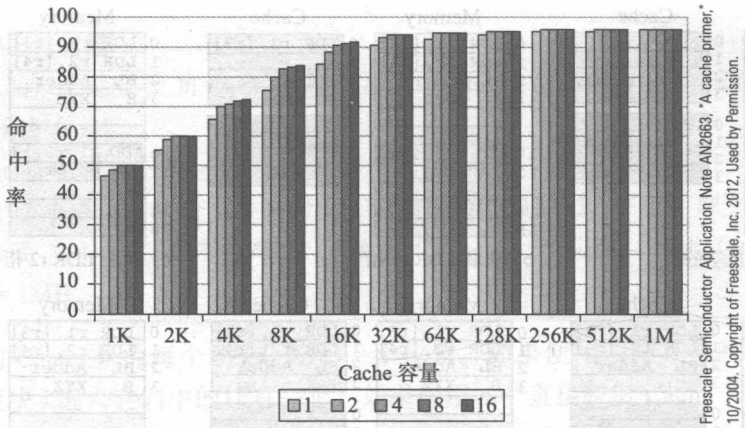


图 1-17 组相联度和 Cache 容量

失效的种类

计算 Cache 系统的效率时，人们通常对命中率感兴趣，因为命中率高使得 Cache 更有效。在设计或改善 Cache 系统时，人们会对失效率而不是命中率感兴趣，因为命中只有一种可能（数据在 Cache 中），而失效却有很多原因。可以通过不断询问“为什么导致失效的数据不在 Cache 中”来提高 Cache 的性能。

Cache 的失效可以分为 3 类：强制（compulsory）失效、容量（capacity）失效和冲突（conflict）失效。强制失效指的是第一次访问数据块时，它一定不在 Cache 中，这是无法避免的。有些处理器通过预测来避免强制失效，在需要的数据被访问之前将其纳入 Cache。当最终访问数据时，由于数据已经在 Cache 中，故第一次访问也不会出现强制失效。这是预取机制的一种形式。

第二种失效为容量失效。在这种情况下，失效的产生是由于工作集（即当前程序所使用的块）比 Cache 容量要大，所需的数据无法全部驻留在 Cache 中。考虑到程序的初始执行。由于 Cache 是空的，所有的第一次失效都是强制失效，每当访问以前没有被缓存的数据，这些新的数据就会被缓存起来。如果程序足够大，当 Cache 被充满，下一次访问就会导致容量失效。此时，系统需要将旧的数据替换出去，以便为缓存新的数据提供空间。

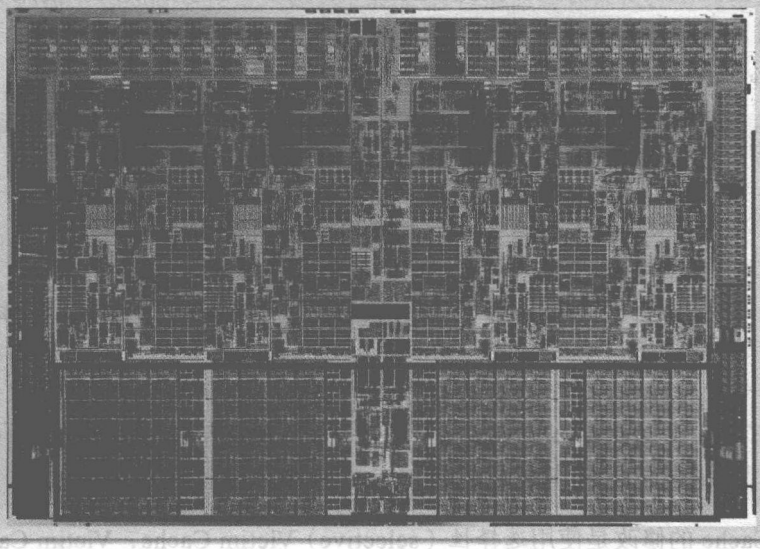
第三种失效为冲突失效。这是最浪费的一种失效，这是因为在 Cache 还未存满时就会发生，此时 Cache 组织形式的副作用导致新的数据与 Cache 中的数据发生冲突。当 m 路组相联 Cache 中全部 m 个组中第 i 块都被占用，新的第 i 块需要被缓存时就会发生冲突失效。在直接映射 Cache 中，冲突失效占全部失效的比例为 20% ～ 40%。全相联 Cache 的冲突失效最小，因为数据块可以被存放在 Cache 中的任意位置。

Cache 污染

Cache 保存了经常需要使用的数据，从而避免经常访问相对慢的主存储器。有时，访问将导致失效，某一块被替换，一个新的块将被载入。该块可能再也不会被访问，但它占用了可以保存其他经常需要访问块的空间。通常称这种问题为 Cache 污染。

Cache 被认为是重要的

下面的照片是 Intel Core i7 的裸片。该处理器由 4 个独立的处理单元构成。照片底部用长方形框出的是 4 个处理单元共享的三级 Cache。展示这张照片是为了说明芯片制造商认为 Cache 是处理器非常重要的组成部分。照片中用于 Cache 的芯片面积要大于单个处理单元的芯片面积。



Reprinted with permission of Intel Corporation.

1.3.4 伪相联、Victim、Annex 和 Trace Cache

由于 Cache 非常重要，针对它的改进已经进行了大量的研究，特别是针对异常行为的处理（例如，当数据被缓存、替换并再次被缓存，等等）。

在直接映射 Cache 基础上进行变更可以得到*伪相联*（pseudo-associative）Cache。当直接映射 Cache 发生冲突失效时，为失效的块提供一个*后备位置*（alternative accommodation）。下面文本框中的“流缓冲和条缓冲”部分对冲突失效和其他类型失效的自然属性进行了进一步的讨论。当直接映射 Cache 发生失效时，伪相联 Cache 根据旧的地址再计算一个新的地址来存放数据。通常情况下，新的地址是对当前地址高端的一位或几位进行取反操作得到。虽然这是规避直接映射 Cache 限制一种巧妙方法，但它需要在一次失效以后进行第二次 Cache 访问。

Victim Cache 是一个小的 Cache，用来保存最近被替换出 Cache 的项目（即*遇难者*（victim））。Victim Cache 与原 Cache 并行访问，理想情况下，它是全相联的。因为它的容量非常小，可以同时搜索的块数有限，因此可以使用全相联的方式构建。

一个小的 Victim Cache 可以减少直接映射 Cache 的冲突失效，这是因为它保存的被替换块的块号与刚调入块的块号相同。Victim Cache 也可以在主 Cache 被充满、开始出现容量失效的时候使用。Victim Cache 保存了被替换出 Cache 的块，由于数据既没有在主 Cache 也没有在 Victim Cache 中复制，因此并没有浪费空间。

Victim Cache 典型应用的例子是嵌套循环[⊖]。考虑在一个循环内调用了一段程序，循环的

⊖ Norman P. Jouppi, “Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers”, WRL Technical Note TN-14, Digital Western Research Laboratory.

起点与被调用程序间有一定的距离。循环开始后，程序被调用。Cache 被充满了，就必须替换一些块为新的指令提供空间。当程序执行完返回循环时，Cache 又需要替换一些块，依此类推。Victim Cache 可以保存被替换的指令并确保它们在循环和函数调用序列被执行时命中。Jouppi 针对 Victim Cache 的研究表明，它可以很小，但十分有效。一些基准测试表明，当使用容量少到 4 项的 Victim Cache 时，可以减少 80% 的冲突失效。Jouppi 在研究使用 Victim Cache 来减少总的失效时，发现不同的基准测试程序的结果差异很大。使用容量为 4 项的 Victim Cache 时，各种基准测试程序平均会使失效率降低 15%，而其中的某个基准测试程序最多会降低 70%。

流缓冲和条缓冲

另一种缓存机制是流缓冲（stream buffer），它利用了相邻块的局部性原理。当失效发生时，流缓冲预取访问块后的若干块。如果处理器稍后访问该数据，就可以从流缓冲中获得。流缓冲在访问数据之前预取数据，从而减少了强制失效。

流缓冲的开销是由于预取可能并不需要的数据从而增加了存储器总线负载。

条缓冲（stride buffer）是一种类似的机制，它利用了诸如数组这种数据结构的特性。在有序的数据结构中，下一个元素很可能就是从当前元素位置加上一个固定偏移量获取（例如，地址 X，X+8，X+16，X+24，……）。条缓冲的操作就是在当前数据的基础上根据合适的偏移量预取下一个数据。

对 Victim Cache 的修改是使用选择性（selective）Victim Cache，Victim Cache 中的项可能来自 Cache 中被替换的块或者主存储器。当新的块被预取时，采用一种预测算法来确定它是否应该被载入 Cache 或是 Victim Cache。预测的目的是为了尽量避免 Cache 污染，即避免载入不会被使用的块。预测机制要求记录每块的历史状态信息。该机制使用了两位指示器：一位用来表示位于 Cache 中的块从来没有被访问过，另一位表示惯性（inertia）以避免在 Cache 和 Victim Cache 间进行过度的交换。

另一个由 John 和 Subramanian[⊖]提出的特殊 Cache 为 Annex Cache。与 Victim Cache 一样，Annex 是一个专用的 Cache，但位于一级 Cache 的前面。即 Victim Cache 位于 Cache 的出口，而 Annex Cache 位于入口。Victim Cache 给替换出 Cache 的数据第二次机会，而 Annex Cache 要求想进入 Cache 的数据证明自己的价值。

Annex Cache 有助于防止很少使用的数据进入 Cache，从而减少 Cache 污染。如果 Cache 为不常使用的数据提供空间而将频繁使用的数据替换出去的话，其效率肯定不高。在启动时，所有数据块以正常的方式进入 Cache。最初阶段后，所有进入 Cache 的数据都需要通过 Annex。只有在主 Cache 发生冲突失效且该失效块已经在 Annex Cache 中被访问了两次时，Annex Cache 中的数据块才会被交换进入主 Cache。进入 Annex Cache 的数据必须被证明在时间或空间局部性上存在价值。在实践中，类似 Annex Cache 这样的复杂 Cache 机制是不容易实现的，由于附加功能的复杂性，需要在本来简单的组相联 Cache 的基础上增加大量的控制电路。

⊖ L. John, A. Subramanian, Annex cache: "A cache assist to implement selective caching." *Microprocessors and Microsystems*, Vol. 23, Issues 8-9, December 1999.

Trace Cache

Trace Cache 是由 Intel 公司设计的专用 Cache。Trace Cache 首先出现在 IA32 系列的 Pentium 4 处理器中。事实上，Pentium 4 的 Trace Cache（取代了一级 Cache）只有 8KB，与之相对比，早期的 Pentium Ⅲ的一级 Cache 有 16KB。

Trace Cache 比传统的 Cache 更进一步，并且可以对指令译码；即它存储经过译码的指令。因此，Trace Cache 不仅可以起到 Cache 的作用，还可以通过提前进行指令译码而减少执行时间。Pentium 4 的 Trace Cache 可以支持每两个时钟周期处理 6 个微操作的流水线。

Pentium 4 的 Trace Cache 具有特殊的功能，用于处理 IA32 复杂的多字长指令体系结构。也可以把 Trace Cache 认为是一种智能 Cache。IA32 和 68000 指令体系结构都支持超长指令字，它们必须被译码成数十甚至数百个微操作。由于不会在 Trace Cache 中装入太多的这种长指令，Intel 提供了一个简单的解决方案——对复杂操作码进行译码，并将其微操作存储在 ROM 中。当遇到长指令时，微操作不缓存。相反，将对微代码 ROM 中恰当的过程调用插入 Trace Cache。

Pentium 4 的 Trace Cache 还包括有限数量的分支预测逻辑，其操作涵盖全部的 Trace Cache，并且与 Pentium 正常的前端分支目标地址预测器相互独立。Trace Cache 分支预测器使用译码的分支指令并在预测的目的地址处获取被译码的微操作（假设它们在 Trace Cache 中）。

1.4 Cache 设计中要考虑的因素

前面已经说过，由于需要考虑的因素很多，Cache 的设计比较复杂，其中一些因素依赖于计算机系统自身的属性。在本节中将讨论一些影响 Cache 系统设计的因素。

存储器管理

在本章的后面将讨论存储器管理。在这里只进行简要的概述。

虚拟存储器是一种管理机制，它在处理器需要数据时将其从磁盘装载到主存储器中。由存储器管理单元（memory management unit，MMU）将 CPU 产生的地址转换为相应的主存储器中的操作数地址。在具有存储器管理的计算机中，操作数的逻辑地址由计算机生成并被转换成主存储器使用的物理地址。由于来自 CPU 的地址被转换成主存储器中数据的位置，因此可以不改变代码实现存储器间数据的移动，只要通过 MMU 改变逻辑地址到物理地址的映射方法即可。

1.4.1 物理 Cache 和逻辑 Cache

在具有存储器管理单元的计算机系统中，Cache 可以位于 CPU 和 MMU 之间，也可以位于 MMU 和物理内存之间。图 1-18 显示了这两种情况。如果在 CPU 数据终端上的数据被缓存，该数据称为逻辑数据（logical data），相应的 Cache 为逻辑 Cache（logical Cache）。然而，如果数据经过 MMU 实现地址转换后被缓存，则该数据称为物理数据（physical data），相应的 Cache 为物理 Cache（physical Cache）。下面将讨论逻辑 Cache 和物理 Cache 的含义以及它们之间的权衡。

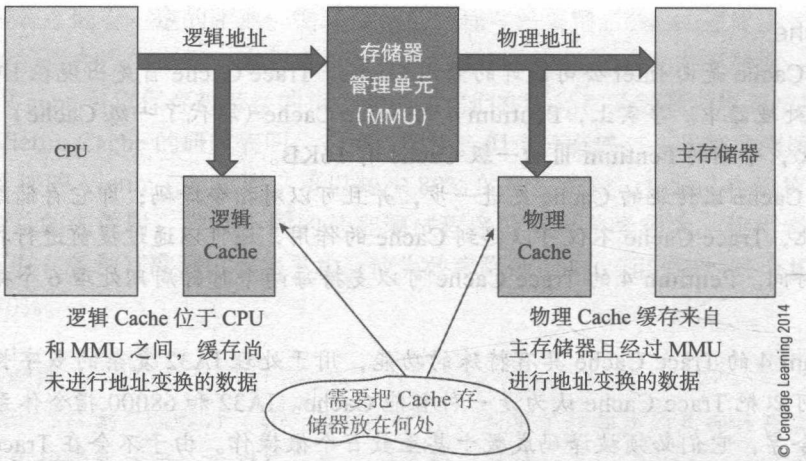


图 1-18 Cache 的位置

物理 Cache 比逻辑 Cache 具有更长的访问时间，这是因为在物理 Cache 中直到 MMU 进行了逻辑地址到物理地址的转换后才可以进行数据访问。逻辑 Cache 比物理 Cache 要快，是因为逻辑 Cache 中的数据可以立刻被访问，而不必等待地址转换。

假设在多任务系统中发生了上下文切换且要执行一个新任务。当新任务开始时，操作系统将相应的地址转换表装入 MMU。当逻辑地址到物理地址的映射关系改变时，Cache 中数据以及相应的物理数据间的关系被打破；不能使用 Cache 中的数据，且逻辑 Cache 必须刷新。物理 Cache 在上下文切换时就不需要刷新。

然而，使用物理 Cache 的代价是在存储器访问前需要额外的时间来执行逻辑地址到物理地址的转换。在实践中，如果把 Cache 页面大小设置为与存储器页面大小一样，就可以实现 Cache 中的块查找与虚拟地址变换并行工作。微处理器一般使用物理 Cache 来减少切换上下文之后导致对 Cache 的刷新。

1.4.2 Cache 电气特性

本书将在下一章中介绍主存储器系统，这里先对 Cache 的电路设计进行简要说明。半导体随机访问存储器有两大类：*静态*（static）和*动态*（dynamic）。静态存储器利用传统数字逻辑将一位数据存储在一个触发器中——正如在《计算机组成原理》第 2 章中描述的那样。静态存储器的特点是低功耗、高速和只要有电就能够保留数据。因此，Cache 通常由静态 RAM 构造。不幸的是，它需要 6 个晶体管来存储一个比特位，因此，静态存储器单元的物理尺寸（即所占硅片的面积）比 DRAM 单元要大得多。这意味着，静态存储器比 DRAM 更昂贵且容量较小，因此不能用来构造大容量的 Cache。

动态存储器（DRAM）通过对单个晶体管的电容充上电荷来保存数据。这使得 DRAM 非常便宜和紧凑，可以用来构造大容量存储器。不幸的是，DRAM 需要更多的电量进行控制，且电容中的电量在几 ms 内就会漏光。为了保存 DRAM 中的数据，必须每隔 4ms 左右定期读取存储单元中的数据并将其写回。DRAM 不适于构建 Cache。

1.4.3 Cache 一致性

Cache 中的数据也在主存储器中。当处理器在写周期内修改数据元素时，必须对主存储

器和 Cache 中的数据副本进行修改，虽然这种修改不必是同时的。因此，可能会出现一个数据存在两个不同副本的情况。如果 Cache 中的数据被修改，而主存储器的数据没有被修改（或倒过来），旧的、没有被修改的数据称为过时的（stale）数据。读者可以想象，这种情况可能会导致严重的错误。假设 I/O 控制器使用 DMA 试图从主存储器移动一块数据到磁盘上，此时处理器刚刚更新了其 Cache 中的数据但尚未修改主存储器中的数据副本。I/O 控制器将把过时的数据而不是把 Cache 中最新的数据从主存储器移动到磁盘上。

Cache 一致性（cache consistency）有时被称为数据一致性（data consistency）。图 1-19 中给出了这样一个系统，两个处理器共享一个存储器。假设在此多处理器系统中的处理器 1 执行了存储器写操作，只更新了自己的局部 Cache 而没有写入存储器。Cache 中的存储器中的数据拷贝是不一致的。这种情况将持续到存储器写回（write back）操作将数据更新。如果处理器 2 在数据更新前读取存储器中相同位置的数据，它从存储器中访问到的就是过时的数据。

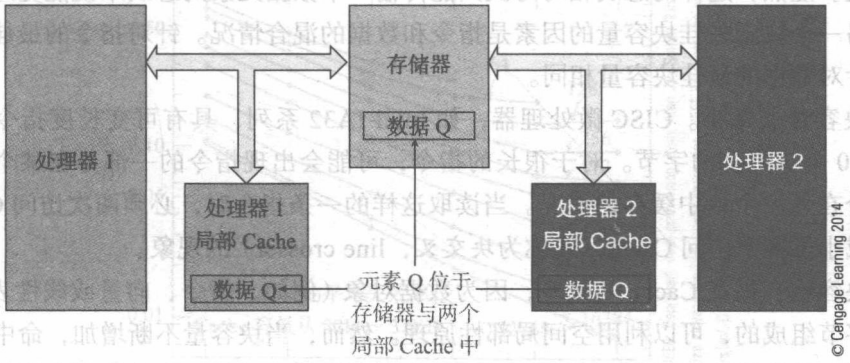


图 1-19 Cache 一致性问题

当多个处理器都有自己的局部 Cache 时也会发生类似的问题。假设处理器 X 同时更新了自己的局部 Cache 和共有的存储器。处理器 Y 也可能在其局部 Cache 中缓存了相同数据的一个副本。处理器 Y 并不知道它缓存的数据已经是过时的。Cache 一致性这个术语意味着在不同的 Cache 和主存储器中的数据必须保持同步（即没有过时的数据）。使 Cache 和主存储器中的数据保持一致（即保证 Cache 一致性）是设计多处理器系统时需要主要考虑的问题之一。

有些处理器通过一种被称为总线侦听（bus snooping）的技术来保持 Cache 的一致性。处理器通过监视地址流和数据流发现那些向主存储器的写访问、同时在自己的 Cache 中也有一个副本的情况。当主存储器中的数据被修改，该处理器局部 Cache 中的内容可以被标记为无效，或将其 Cache 更新。

1.4.4 块大小

块是 Cache 中的基本存储单位。一个重要的问题是，要多大的块才能获得最佳性能？针对块大小和 Cache 性能之间的关系已经展开了大量的研究，有时是通过模拟软件的 Cache 操作，有时是通过监控计算机系统中真实 Cache 的操作。

失效率

前面已经谈到了 Cache 的命中率（hit ratio）h，表示访问操作在 Cache 中发现数据的比例。典型的命中率在 0.90 ～ 0.98 之间。

失效率代表访问操作没有在 Cache 中发现数据的比例，由 $m = 1-h$ 表示。显然，无论使用 h 还是 m 都是可行的。但是，当比较两个高性能 Cache 时，h 的值可以是 0.98 或 0.99，差异仅有 1%。但是，从失效率的角度看，m 的值分别是 0.02 和 0.01，它们之间有 2 倍的差异。因此，计算机科学家们发现使用 m 可以带来更多的启发。

最佳的块容量取决于几个方面，尤其是正在执行的程序的性质。负责控制处理器和存储器之间数据流量的总线协议也会影响 Cache 的性能。典型计算机的总线负责将地址传送到主存储器，然后从数据总线发送或接收一个数据字——每次存储器访问都需要一个地址和一个数据元素。假设总线可以工作在突发模式（burst mode），即发送一个地址，然后获得一批连续的数据值。显然，这样的总线相对每次只能传输一个数据元素的总线来说能更好地传输较大的块。另一个决定最佳块容量的因素是指令和数据的混合情况。针对指令的最佳块容量并不一定与针对数据的最佳块容量相同。

假设块容量非常小。CISC 微处理器，如 Intel IA32 系列，具有可变长度指令，其范围从 2 个到 10 个或更多的字节。对于很长的指令，可能会出现指令的一部分在某个块中缓存而另一部分在另一个块中缓存的情况。当读取这样的一条指令时，必须两次访问 Cache。增加块大小减小了多次访问 Cache（或称为块交叉，line crosser）的现象。

增加块容量会提高 Cache 的效率，因为数据对象（例如，指令、向量或线性表）是由一组连续的字节组成的，可以利用空间局部性原理。然而，当块容量不断增加，命中率也会下降，因为减少了块的数量使得给定对象被缓存的概率降低。此外，大的块的性能很大程度上依赖于数据访问的局部性。当发生失效将一块调入 Cache 时，它可能包含不经常访问的数据，反而把经常访问的数据替换出去了。

Cache 与程序员

代码的编写方式对 Cache 的性能有很大的影响。在一个 C 数组中，元素以行的方式顺序存储。例如 x[0,0] 的下一个数据是 x[0,1]。以行的顺序访问元素可以充分利用空间局部性，因为元素 x[i,j] 将导致 x[i,j+1] 被缓存，前提是它们在同一块中。每次以列的方式访问元素时都可能会产生失效，这是因为连续两个被访问的数据不在同一块中。

图 1-20 给出了数据访问的块容量和 Cache 容量之间的关系。这些经典的结果是通过模拟得到的，当时 Cache 的容量比现在的要小得多。图 1-20 画出的是失效率而不是命中率的情况。每条曲线对应一个特定的 Cache 容量（从 32B ～ 32KB）。图 1-21 提供指令 Cache 的相应结果。数据 Cache 的失效率首先逐渐变好（即越来越小）然后逐渐恶化（即越来越大）直到块容量与 Cache 容量本身相等；而指令 Cache 的失效率随块容量的增加而减小。这表明，访问的局部性对指令的作用比对数据的作用更强。一般情况下，只要给定 Cache 容量就有一个最佳的块容量；例如 256B 的 Cache，最佳块容量为 64B。Cache 容量越大，最佳的块容量就越大。

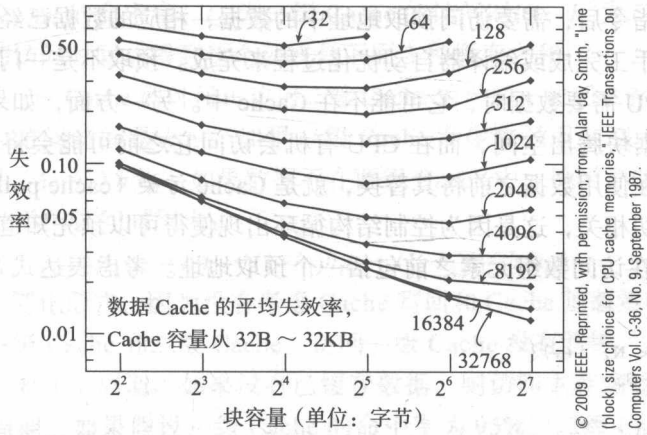


图 1-20 数据 Cache 的平均失效率与块容量的关系（每条曲线对应给定的 Cache 容量）

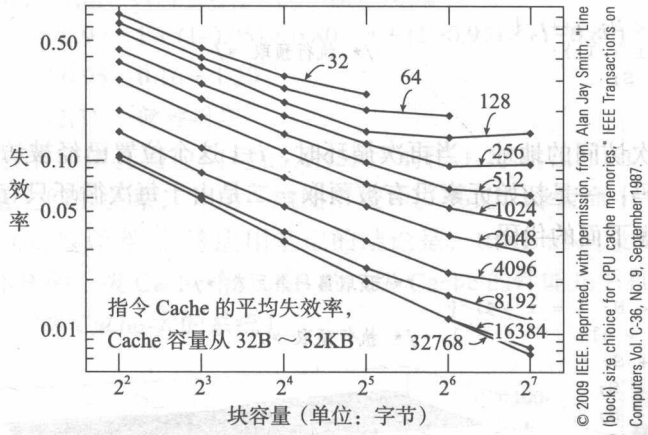


图 1-21 指令 Cache 的平均失效率与块容量的关系（每条曲线对应给定的 Cache 容量）

1.4.5 取指策略

有几种策略可以用于降低失效率，从而提升 Cache 性能（例如，按需获取、预取、选择性获取）[⊖]。按需获取（demand fetch）策略在失效后调入所需块，这是一种最简单的选择。预取（prefetch）策略预测未来 Cache 的需求（例如，如果没有缓存块 i+1，当访问块 i 时也调入第 i+1 块）。实现预取算法有许多可能的方法。选择性获取（selective fetch）策略在主存储器的部分内容不能被缓存的情况下使用。例如，在多处理器系统中，由多个处理器共享的数据就不应该被缓存，如果这些数据被缓存而处理器修改了存储器中的拷贝，Cache 和存储器中的数据将不再保持一致。

如果需要快速访问数据，就应该尽早地获取它。通过预取（prefetch）数据可以最大限度地发挥 Cache 的作用。一些微处理器的指令集包括预取指令，它只产生操作数地址而不做其他事情。当操作数出现在总线上时，Cache 系统自动缓存这个地址上的数据。该指令并没有其他功能，只是一个触发预取的虚拟操作。

⊖ S.P. Vander Wiel and D.J. Lilja, "When caches aren't enough: data prefetching techniques," *Computer*, July 1997, pp.23-30.

如果，在几个指令后，需要访问预取地址中的数据，相应的数据已经在 Cache 中。该预取操作可由程序员手工完成或编译器自动优化过程来完成。预取不是一门精确的科学。如果预取得太晚，当 CPU 需要数据时，它可能不在 Cache 中。另一方面，如果数据预取得太早，Cache 要为新的数据块腾出空间，而在 CPU 有机会访问它之前可能会将其替换出去。这样过早地预取数据又在使用数据之前将其替换，就是 Cache 污染（cache pollution）的例子。

预取与循环密切相关，这是因为控制结构循环出现使得可以预先知道将使用的数据。预取最简单的方式是在访问数组元素之前包括一个预取地址。考虑表达式 $S = \sum_{ai}$ 的计算。相应的代码是：

```
for (i = 0; i < N; i++){
    S = a[i] + S;
}
```

根据 Wiel 和 Lilja 给出的例子[⊖]，这里使用 `fetch(&address)` 来表示预取操作和预取的地址。预取最简单的例子是在循环中使用地址前访问该地址；即

```
for (i = 0; i < N; i++) {
    fetch (&a[i + 1]);                /* 执行预取 */
    S = a[i] + S;
}
```

这样将产生下次访问的地址，当再次循环时，i+1 这个位置已经被访问。可以在下面两个方面改进该段代码：一是初始元素没有被预取；二是由于每次循环只有一个有效操作所以循环效率较低。考虑下面的代码：

```
   fetch (&a[0]);                 /* 预取第一个元素 */
for (i = 0; i < N; i = i +4) {
   fetch (&a[i + 4];              /* 执行预取 */
   S = a[i]   + S;
   S = a[i+1] + S;
   S = a[i+2] + S;
   S = a[i+3] + S;
}
```

在这种情况下，一次循环执行 4 个操作。由于每次取指调入 Cache 中的数据块中所包含 16 个字节可以被 4 个连续的指令使用，因此只需要做一次预取。

1.4.6 多级 Cache

在 20 世纪 90 年代后期，存储器价格暴跌，半导体技术可以将非常复杂的系统放在芯片上，时钟频率达到了 500MHz（时钟周期时间只有 2ns）。Cache 系统的容量和复杂性都增加了，计算机系统开始实现两级 Cache：在 CPU 内部的一级 Cache 和在主板上的二级 Cache。两级 Cache 系统中使用少量速度非常高的一级 Cache 和由大量速度快的存储器构成的二级 Cache。换句话说，有两个层次的 Cache 串在一起。首先访问速度快、容量小的一级 Cache。如果没有所需数据，则访问速度较慢、容量较大的二级 Cache。如果仍然没有找到数据，则需要访问主存储器。两级 Cache 系统的访问时间由一级 Cache 的访问时间加上二级 Cache 的访问时间再加上主存储器的访问时间；即

$$t_{ave} = h_1 t_{c1} + (1-h_1)h_2 t_{c2} + (1-h_1)(1-h_2)t_m$$

⊖ S.P. Vander Wiel and D.J. Lilja, "When caches aren't enough: data prefetching techniques," *Computer*, July 1997, pp.23-30.

其中，h_1 为一级 Cache 的命中率；t_{c1} 为一级 Cache 的访问时间；h_2 为二级 Cache 的命中率，t_{c2} 为二级 Cache 的访问时间。得到这个公式是下面 3 种概率之和：

t_{ave} = 访问一级 Cache 的时间 + 访问二级 Cache 的时间 + 访问主存储器的时间

一级 Cache 的访问时间是 h_1t_{c1}。如果一级 Cache 发生失效且二级 Cache 命中，访问二级 Cache 的时间是（$1-h_1$）h_2t_{c2}。如果数据不在两级 Cache 中，访问存储器的时间为（$1-h_1$）（$1-h_2$）t_m。因此，总的访问时间为：

$$t_{ave} = h_1t_{c1} + (1-h_1)h_2t_{c2} + (1-h_1)(1-h_2)t_m$$

该公式为一个简化形式，因为没有考虑 Cache 写回和 Cache 加载策略。考虑下面这个例子。某计算机有一级 Cache 和二级 Cache。访问一级 Cache 没有开销，需要 1 个周期。在二级 Cache 中命中需要 4 个周期。如果没有已缓存数据，则访问主存储器，包括 Cache 加载，需要 120 个时钟周期。如果假设一级 Cache 的命中率为 95%，二级 Cache 的后续命中率是 80%，平均访问时间是多少？

$$\begin{aligned} t_{ave} &= h_1t_{c1} + (1-h_1)h_2t_{c2} + (1-h_1)(1-h_2)t_m \\ &= 0.95\times1 + (1\text{-}0.95)\times0.80\times4 + (1-0.95)\times(1-0.80)\times120 \\ &= 0.95 + 0.16 + 1.20 \\ &= 2.31 \text{ 个时钟周期} \end{aligned}$$

来自飞思卡尔半导体公司应用笔记 AN2663 的图 1-22 给出了命中率与一级 Cache 及二级 Cache 的容量之间的关系，并用三维图形表示。可见，峰值命中率是 96%，此时执行了特定的代码（GCC 编译器）。该应用笔记的结论是，16KB 的一级 Cache 和 1KB 的二级 Cache 的性能与 1KB 的一级 Cache 和 16KB 的二级 Cache 的性能几乎相同（虽然没有人会设计出一级 Cache 比二级 Cache 大的系统）。

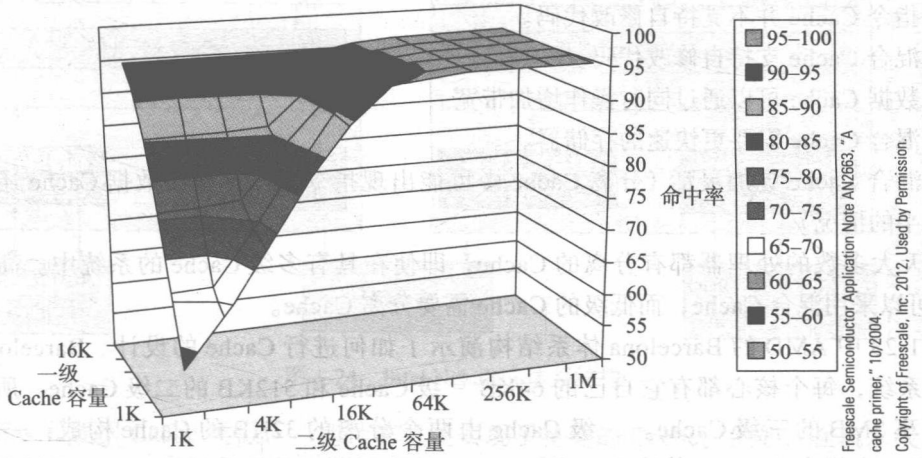


图 1-22 命中率与一级 Cache 及二级 Cache 的容量

自修改代码

早期计算机的指令集体系结构是很原始的，程序员需要利用冯·诺依曼机器来编写自修改代码。顾名思义，代码可以在运行时动态变化。

例如，如果计算机没有索引，可以编写如下代码：

```
100 LOAD r0,2325
101 ADD  100,#1
```

第一行将存储位置2325的内容载入r0。第二行对指令100的内容增加1，即对上一条指令本身加1。如果操作数地址在指令位置的后部，它将变成2326，这就是自修改代码。

自修改代码难以阅读和调试，大多数程序员都不使用它。此外，它往往不能在具有Cache或存储器管理的环境中工作，因为无法直接访问存储器中的代码。

1.4.7 指令和数据 Cache

数据和指令都是冯·诺依曼概念的核心；即它们共享相同的存储器。Cache设计者可以选择使用混合Cache（unified cache）来缓存数据和指令，或者实现分离的数据和指令Cache（split cache）。将数据和指令Cache分开是很有意义的，因为它们有不同的特性。除了将块调入Cache以外，是不会修改指令Cache中的项目的。此外，不用担心修改那些替换出指令Cache的指令，这是因为程序在其执行过程中不发会变化。由于指令Cache中的内容不会被修改，实现指令Cache要比实现数据Cache容易得多。将指令和数据Cache分别实现可以提高CPU与存储器间的带宽，这是因为指令和数据可以同时读取。对于流水线系统来说，为了实现指令和数据的同时访问，必须采用分离的指令和数据Cache。混合Cache和分离Cache的特点如下：

- 指令Cache可以为产生指令流而优化。
- 数据Cache可以为读写操作而优化。
- 数据Cache可以单独进行优化。
- 指令Cache并不支持自修改代码。
- 混合Cache支持自修改代码。
- 数据Cache可以通过同时操作增加带宽。
- 混合Cache需要更快速的存储器。
- 混合Cache更加灵活（分离Cache中可能出现指令Cache满而数据Cache还空着一半的情况）。

今天大多数的处理器都有分离的Cache，即使在具有多级Cache的系统中。高级别的Cache可以采用混合Cache，而低级的Cache需要分离Cache。

图1-23中AMD的Barcelona体系结构演示了如何进行Cache的设计。Barcelona是一个多核系统，每个核心都有它自己的64KB一级Cache和512KB的二级Cache。所有4个核心共享2MB的三级Cache。一级Cache由两个分离的32KB的Cache构成：一个数据Cache和一个指令Cache。传统上，多级Cache的访问顺序是从最低级Cache开始根据失效情况逐渐延伸到更高级的存储层次（先访问一级Cache，然后询问二级Cache，接着是三级Cache、主存储器，直到失效的数据被找到）。在Barcelona体系结构中，一级Cache是所有Cache加载的最终目标，所有取得的数据都放到一级Cache中。二级Cache保存被替换出一级Cache中的数据。因为一级Cache和二级Cache之间是紧耦合的，所以从二级Cache到一级Cache传输数据所产生的延迟较低。

三级Cache被多个核共享。加载数据时将直接从三级Cache传给一级Cache而不通过二级Cache。被传送的数据要么由于需要被多个处理器使用而保持在三级Cache中，要么就

因为不需要共享而删除。与二级 Cache 类似，三级 Cache 不保存来自存储器的数据，而是保存被替换出二级 Cache 的数据。

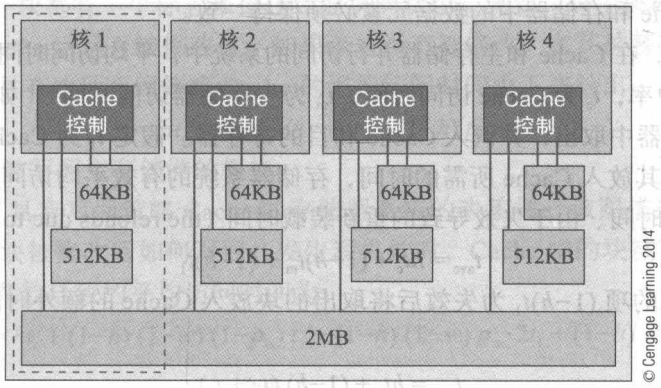


图 1-23 AMD Barcelona 体系结构

图 1-24 给出了与 Barcelona 同期的 Intel Nehalem 架构[⊖]。它的一级 Cache、二级 Cache 和三级 Cache 的大小分别是 32K、256K 和 8M 字节。

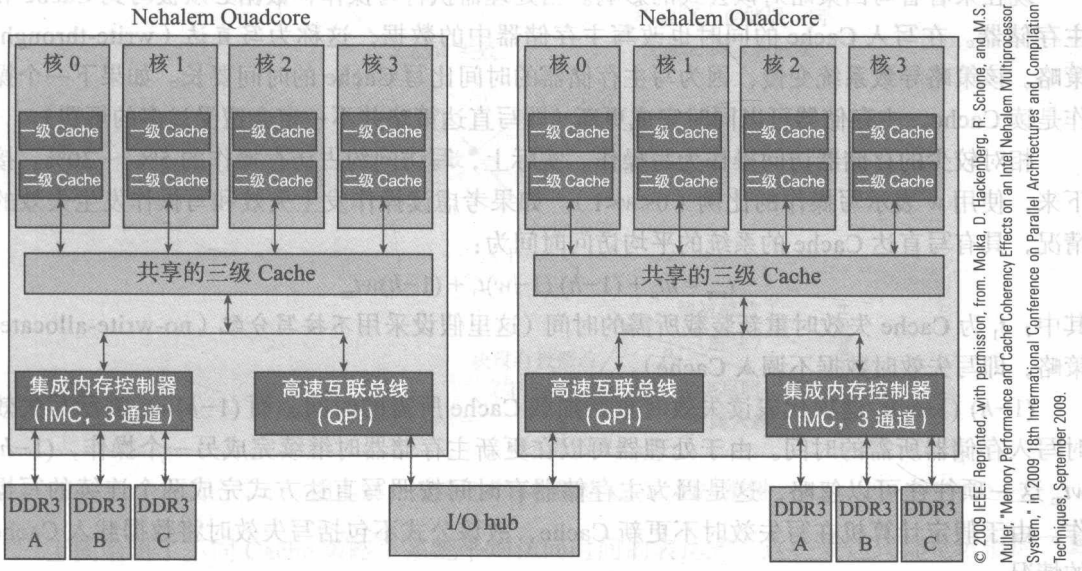


图 1-24 Intel Nehalem 体系结构

与指令和数据 Cache 类似，一些计算机还实现了特殊的 Cache。例如，在流水线中引入分支目标 Cache（branch target cache）。分支目标 Cache 存储与分支有关的信息，例如分支地址和目标地址处的指令操作码。同样，可以在特殊的返回地址 Cache 中保存子程序返回地址，以减少当返回地址保存在堆栈中时从子程序返回的开销。

1.4.8 写 Cache

到目前为止，本章只考虑了对 Cache 的读访问（访问最频繁的形式）。现在来看看更复

⊖ Trent Rolf, "Cache organization and memory management of the Intel Nehalem computer architecture," University of Utah, http://rolfied.com/nehalem/nehalemPaper.pdf.

杂的写访问。当处理器写数据到 Cache 时，在 Cache 和主存储器中的相应块都需要修改，虽然这两个操作不需要同时执行。然而，必须在下一次访问前确保存储器中的数据元素副本已被更新；即在 Cache 和存储器中的数据元素必须保持一致。

前面已经指出，在 Cache 和主存储器并行访问的系统中，平均访问时间是 $t_{ave} = ht_c + (1-h)t_m$（其中，$h$ 为命中率，t_c 为 Cache 访问时间，t_m 为主存储器访问时间）。如果数据不在 Cache 中，它必须从存储器中取出，并调入 Cache 和目的寄存器。假定 t_l 为 Cache 失效时从主存储器中取出一块并将其放入 Cache 所需的时间，存储器系统的有效平均访问时间是 Cache 访问时间、存储器访问时间、由于失效导致的重新装载时间（the reloads due to miss）之和：

$$t_{ave} = ht_c + (1-h)t_m + (1-h)t_l$$

上式中新出现的项 $(1-h)t_l$ 为失效后将取出的块放入 Cache 的额外时间。该表达式可以改写为：

$$t_{ave} = ht_c + (1-h)(t_m + t_l)$$

访问导致失效的元素与将存储器中的块调入 Cache 可以同时进行。因此（$t_l + t_m$）项应该是 $\max(t_l|t_m)$，因为 $t_l>t_m$，上式可以写成：

$$t_{ave} = ht_c + (1-h)t_l$$

现在来看看写回策略对该公式的影响。当处理器执行写操作，数据必须被写到 Cache 和主存储器。在写入 Cache 的同时也改写主存储器中的数据，这称为写直达（write-through）策略。该策略导致系统变慢，因为写主存储器的时间比写 Cache 的时间要长。如果下一个操作是读 Cache，主存储器可以同时完成更新（即写直达策略并不一定会遭受过多的惩罚）。

相对较少的存储器访问操作为写操作。实际上，写访问约占访存操作的 5% ～ 30%。接下来，使用 w 表示写操作的比例（$0<w<1$）。如果考虑读操作发生失效和写操作发生失效的情况，具有写直达 Cache 的系统的平均访问时间为：

$$t_{ave} = ht_c + (1-h)(1-w)t_l + (1-h)wt_m$$

其中：t_l 为 Cache 失效时重新装载所需的时间（这里假设采用不按写分配（no-write-allocate）策略，即写失效时数据不调入 Cache）。

$(1-h)(1-w)t_l$ 这项代表读失效时重新加载 Cache 所需的时间，而 $(1-h)wt_m$ 表示写失效时写入存储器所需的时间。由于处理器可以在更新主存储器时继续完成另一个操作，$(1-h)wt_m$ 这一项往往可以忽略，这是因为主存储器有时间按照写直达方式完成两个连续的写操作。由于假定计算机在写失效时不更新 Cache，故该公式不包括写失效时将数据载入 Cache 的情况。

Cache 的性能可以通过写缓冲（write buffer）来改进，它保存了等待写入存储器的数据。典型的写缓冲保存 4 个地址 / 数据对。当然，必须保证处理器要访问的数据刚被更新、不在存储器而在缓冲区中，此时，处理器可以访问缓冲区。一种解决方案是在执行读操作之前允许写缓冲完成存储器更新。

另一种修改存储器的策略被称为写回（write-back）。在具有写回策略的 Cache 系统中，向主存储器的写操作只有在 Cache 块被替换时才会发生，即对 Cache 的写操作并不会每次都导致对主存储器的更新。只有在某块由于读失效而被替换出去时才将该块写回存储器。因此上式可以写为：

$$\begin{aligned} t_{ave} &= ht_c + (1-h)(1-w)t_l + (1-h)(1-w)t_l \\ &= ht_c + 2(1-h)(1-w)t_l \end{aligned}$$

注意出现了两个 $(1-h)(1-w)t_1$，这是因为读失效导致被替换的块写回存储器，同时新的块被调入 Cache。

Cache 中的每一块都有一个标志来描述当前块的状态。例如，每个块都有一个脏（dirty）位来表示它调入 Cache 后是否被修改过。如果该块没有被修改，在其被替换出 Cache 时就不需要写回存储器。具有这种写回策略 Cache 的平均访问时间由下式给出：

$$t_{ave} = ht_c + (1-h)(1-w)t_1 + (1-h)p_w w t_1$$

其中，p_w 为块需要被写回主存储器的概率。

图 1-25 给出了具有写回策略 Cache 的存储器系统的决策树。该图描述了在读失效时如何修改 Cache 以及块被修改后如何写回。发生写失效时，Cache 中的块被写回，并将新的块装入 Cache。这些参数导致的平均访问时间为：

$$t_{ave} = ht_c + (1-h)(1-w)(1-p_w)t_1 + (1-h)(1-w)p_w \cdot 2t_1 + (1-h)w \cdot 2t_1$$

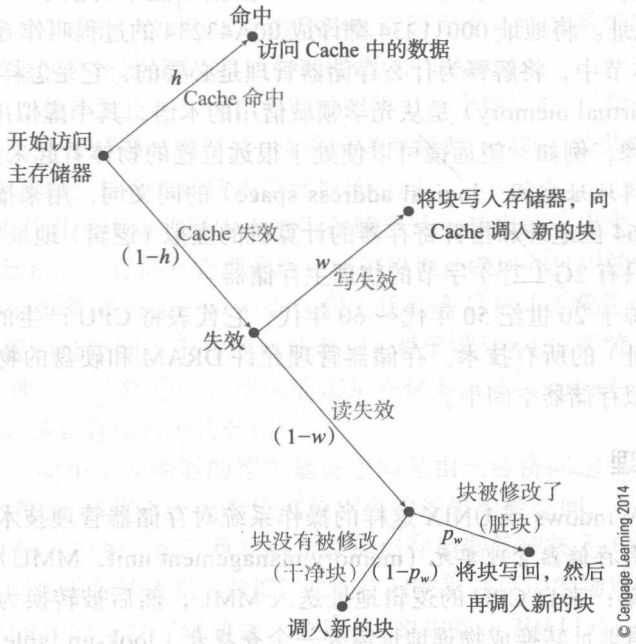


图 1-25 访问具有写回策略 Cache 的决策树

上面给出了不同 Cache 策略下系统平均访问时间的表达式。这些表达式都是近似的，真实系统的行为将取决于其具体实现。

尽早启动、关键字优先和非阻塞 Cache

改善 Cache 性能的一种方法是执行早启动（early restart）。也就是，只要块中所需的字已从存储器中取出，CPU 就可以使用它而不必等到整个块都调入 Cache。

另一种提高性能的方法，称为关键字优先（critical word first），即先从主存储器获取导致失效的那个字，这样处理器可以继续执行，然后再把对应块中其他字读出来成为一块调入 Cache。

非阻塞 Cache（Non-blocking Cache）是另一种试图减少失效影响的方法。通常情况下，当发生失效时，需要向 Cache 调入新块，此时处理器处于暂停状态。然而，有可能

下一次存储器访问并非访问被替换的块。在这种情况下，处理器可以继续执行。使用在第4章中讨论的分离事务总线（split transaction bus）使之成为可能。

1.5 虚拟存储器和存储器管理

存储器管理（Memory Management）是操作系统和硬件的切合点，它关注的是管理主存储器和磁盘。从许多方面看，存储器管理是一种扩展的Cache技术。

计算机刚出现时，计算机生成的地址对应物理或真实存储器中操作数的位置。即使在今天，各种控制器中的8位微处理器通常不使用存储器管理技术。在PC和工作站等高性能计算机上产生的*逻辑地址*（logical address），并不是操作数在存储器中的*物理地址*（physical address）。考虑指令 `LDR r2,[r3]`，其作用是将由r3寄存器指向的存储器位置中的内容送入r2寄存器，假定寄存器r3的内容是0x00011234。数据可能来自基于DRAM的主存储器中0x00A43234这个地址。将地址00011234翻译成00A43234的过程叫作*存储器管理*（memory management）。在本节中，将解释为什么存储器管理是必要的，它是怎样实现的。

虚拟存储器（virtual memory）是从光学领域借用的术语，其中虚拟用于描述一种似乎在某个位置的虚拟图像，例如，望远镜可以使处于很远位置的物体看起来好像就在面前一样。虚拟存储空间是*逻辑地址空间*（logical address space）的同义词，用来描述计算机可以访问的地址空间。具有64位地址和指针寄存器的计算机的虚拟（逻辑）地址空间为2^{64}字节，即使它在系统中可能只有2G（2^{31}）字节的物理主存储器。

存储器管理起源于20世纪50年代～60年代，它代表将CPU产生的逻辑地址转化成实际地址（即物理地址）的所有技术。存储器管理允许DRAM和硬盘的物理地址空间无缝地合并到计算机的虚拟存储器空间中。

1.5.1 存储器管理

计算机使用如Windows或UNIX这样的操作系统对存储器管理技术进行扩展使用。图1-26给出了一个具有*存储器管理单元*（memory management unit，MMU）的系统的结构。原则上，操作非常简单：来自CPU的逻辑地址送入MMU，然后被转换为操作数在存储器中的物理地址。将逻辑地址转换成物理地址需要一个*查找表*（look-up table）。因为将每个逻辑地址都转换成物理地址确实需要非常大的表，因此存储空间通常被分成页（page），每个逻辑页中的地址被转换成物理页中对应的地址。一个页通常为4KB。例如，如果页大小为4KB，处理器有32位地址空间，逻辑地址0xFFFFAC24可能被转换为0x00002C24。

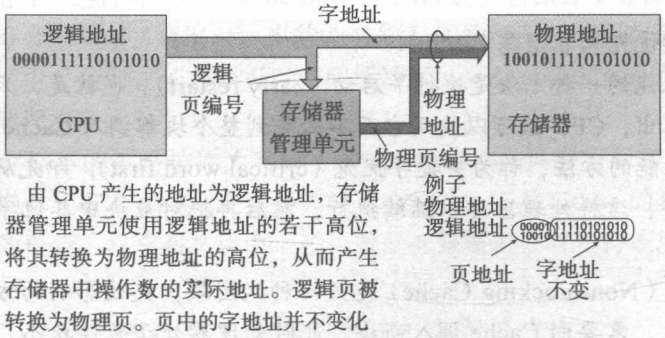


图1-26 存储器管理单元

处理器逻辑地址空间的大小与指定操作数的寻址方式无关。它也与程序是用高级语言、汇编语言还是机器代码编写无关。在 32 位的系统中，指令 `LDR r4,[r6]` 允许使用 4GB 的逻辑空间。无论采用何种技术，处理器不能指定 4GB（范围从 0 ～ $2^{32}-1$）以外的逻辑地址，这仅仅是因为其程序计数器的位数是有限的 32 位。

物理地址空间由处理器使用存储器系统中所有的实际地址位置构成。该存储器不是抽象的，是真正实现的。换句话说，系统的主存储器构成物理地址空间。计算机逻辑地址空间的大小由指定地址所需的位数决定，而物理地址空间的大小通常只受其成本限制。

现在可以看到，为什么微处理器的逻辑和物理地址空间的大小是不同的。更奇怪的是为什么微处理器使用存储器管理，将例如 0x00001234 这样的逻辑地址转换为物理地址 0x861234。存储器管理系统的基本目的如下：

1）实现对物理地址空间大小超过了逻辑地址空间大小这样的系统的控制（例如，某 8 位微处理器具有 16 位地址总线（64KB 的逻辑空间）来访问 2MB 的物理内存）。

2）实现对逻辑地址空间大小超过了物理地址空间大小这样的系统的控制（例如，某 32 位微处理器具有 4GB 的逻辑地址空间，管理 64MB 的 RAM）。

3）存储器保护，包括防止一个用户访问分配给另一个用户的存储空间的机制。

4）存储器共享，一个程序可以与另一个程序共享资源（例如，公共数据区或公共代码）。

5）高效利用存储器，最有效地使用物理地址空间。

6）将程序员从考虑程序和数据应该位于存储器的何处中解放出来。即程序员可以根据其意愿使用任意的地址，而存储器管理系统会将逻辑地址映射到可用的物理地址。

一个真正的存储器管理单元可以不必实现上述所有目标（注意前两个目标是互斥的）。第二个目标（即逻辑地址空间大于物理地址空间）对于设计 64 位系统尤其重要。当针对该问题使用存储器管理时，这就是经常所说的*虚拟存储器技术*（virtual memory technology）。虚拟存储器几乎就是逻辑存储器的代名词。

可用物理地址空间小于处理器的逻辑地址空间是由于经济问题所造成的，并一直困扰着主流企业。20 世纪 50 年代末，大型机可使用大的逻辑地址空间，但只能使用 2K 左右的 RAM。英国曼彻斯特大学的一组计算机科学家提出存储器管理技术（现被称为虚拟存储器）来解决该问题。他们将部分的逻辑（虚拟）地址空间映射到可用的物理地址空间，如图 1-27 所示。该例中，256KB 的一段逻辑地址，范围从 780000 ～ 7BFFFF，被映射到物理存储器范围从 00000 ～ 3FFFF 的区域。只要处理器访问逻辑地址空间中的数据，其地址将被映射到相应的物理地址，一切都很顺利。这里假定从 CPU 发出的地址将直接（即不改变地）交给系统的地址总线。

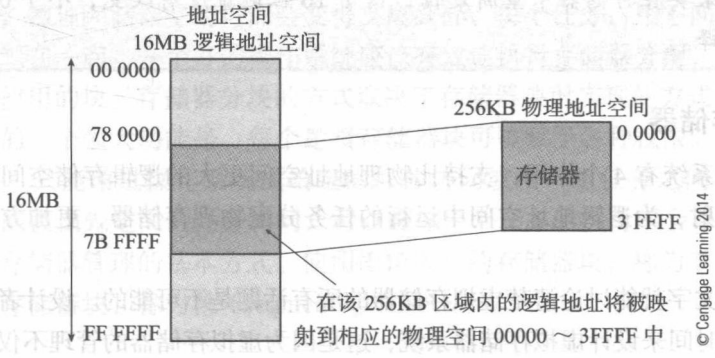


图 1-27 将逻辑地址空间映射到物理地址空间

当处理器产生操作数的逻辑地址不能被映射到可用的物理地址空间时就会出现问题。曼彻斯特大学采用的解决方案非常简单。每当处理器生成没有物理地址对应的逻辑地址时，操作系统中止当前程序的执行并处理该问题。操作系统从磁盘存储器中取出一块包含所需操作数的块，并把该块放在物理存储器中（覆盖所有旧的数据），然后告诉存储器管理单元，逻辑地址空间和物理地址空间之间存在一个新的关系。换句话说，程序或数据被保存在磁盘中，只有当前需要的部分程序被传送到物理 RAM 中。存储器管理单元跟踪由处理器产生数据的逻辑地址与该数据在物理存储器中位置之间的关系。整个过程非常复杂，需要在处理器体系结构、存储器管理单元以及操作系统之间进行协调。人们希望看到简单的虚拟存储器系统，但实际这是一场噩梦。

两个处理器的故事

第一代处理器（非更早的 Intel 4004）有 16 位的寄存器和地址总线，提供了 2^{16}=64KB 的地址空间。虽然只可以访问 64KB 的存储器，但在 20 世纪 70 年代后期，它还是比较大的存储器，处理器应用一般只使用比 64KB 少的物理存储器。

当 Motorola 推出了 68000 处理器，其地址寄存器为 32 位，提供 2^{32} 字节的地址空间。该地址空间是线性的、连续的，不能以任何方式划分。然而，为了降低制造成本，68000 只有 24 个地址引脚，具有 2^{24}=16MB 的物理地址空间。十六进制的 32 位地址表示为 $XXYYYYYY_{16}$，其中 YYYYYY 表示物理地址，XX 代表 8 个“无所谓”位，这是因为它们不能通过地址总线访问。当时，16MB 的存储空间被认为是非常大的物理地址空间。

Intel 通过其 8 位的 8080 和 16 位 8086 击败 Motorola 而占有了市场。不像 68000 具有 32 位地址寄存器，8086 只有 16 位地址寄存器（包括程序计数器 PC），使逻辑地址空间限制为 64KB。8086 通过被称为分段（segmentation）的技术来访问 2^{20} 字节的存储器。当 8086 产生一个 16 位的地址，它被增加到 20 位，使得新地址可以访问 1MB 的空间。8086 有 4 个分段寄存器（segment registers）允许程序员访问 1MB 物理存储器中的 4 个分段。这些分段为代码段、数据段、扩展段和堆栈段，分别用 CS 段寄存器、DS 段寄存器、ES 段寄存器和 SS 段寄存器指示。

可以将从地址寄存器取出的 16 位地址加上由段寄存器左移 4 位的结果得到 20 位地址；即地址由 R+16S 得到，其中 R 是一个指针，S 为一个段寄存器。虽然相当麻烦，但 Intel 的寻址机制提供了一种实现单独的代码 / 数据 / 堆栈地址空间、并使重定位十分简单的途径。如果要在存储器中重新定位，由于 16 位地址没有改变，小于 64KB 的代码片段不必重新编译。

1.5.2 虚拟存储器

虚拟存储器系统有 4 个作用：支持比物理地址空间更大的逻辑存储空间，实现逻辑地址到物理地址的映射，为逻辑地址空间中运行的任务分配物理存储器，更加方便地建立多任务的系统。

使用有限的文字就能讨论清楚虚拟存储器的所有话题是不可能的。设计者和程序员组成的团队花费很长的时间来设计虚拟存储器系统，这是因为虚拟存储器的管理不仅复杂而且几乎在支持多用户或多任务操作系统的系统中通常都是各不相同的。本节只对虚拟存储器进行概述。

1. 存储器管理和多任务

多任务系统通过周期性地在任务之间进行切换以支持两个或多个进程同时执行。显然，多任务处理只有当几个任务同时驻留在主存储器中时才是可行的。如果任务每次运行都必须从硬盘装载，则切换新任务需要的时间将过长。

图 1-28 演示了具有两个任务的系统中如何将逻辑地址空间映射到物理地址空间。图中，任务 A 和 B 同时驻留在物理存储器中。每个任务都有其自己的逻辑存储空间（例如，程序和堆栈）可以访问位于物理存储空间的共享资源。程序员可以完全自由地为任务的不同组件选择其自身的地址。因此，图 1-28 中，任务 A 和 B 可以访问物理存储器中相同的数据结构，即使它们使用不同的逻辑地址。即每个任务只知道自己与另一个任务共享数据的一个副本。

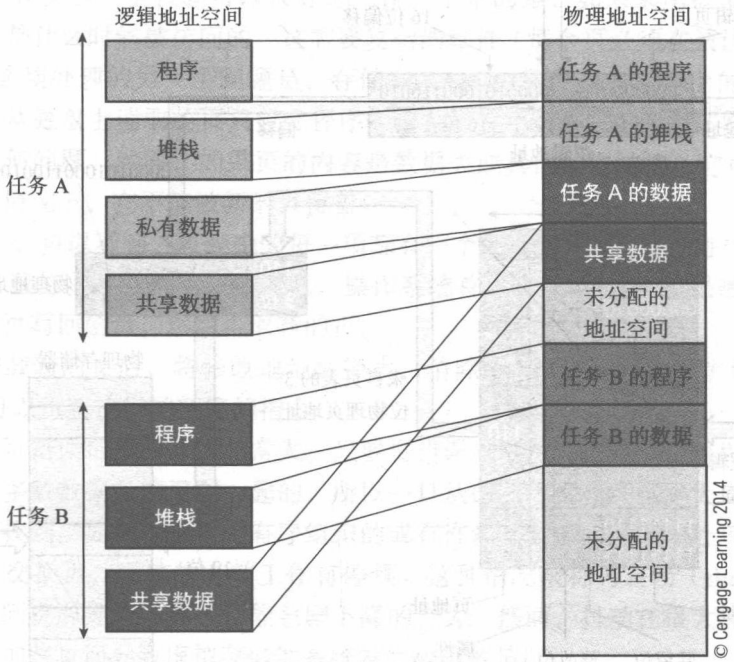


图 1-28　多任务环境中的地址映射

存储器管理单元将程序员选择的逻辑地址映射到物理存储器空间，而操作系统负责建立逻辑地址到物理地址的映射表。当创建一个新的任务时，将告知操作系统任务对存储器的需求。操作系统搜索可用的物理存储器空间，寻找空闲的存储块并将其分配给任务。可以想象，一段时间后，物理存储器空间可能会变得支离破碎，每个任务占用不同的存储块，以非常复杂的方式交织在一起。一个好的操作系统应该有效地执行存储器分配，不允许物理存储器中存在大量未使用的块。存储器分块的方式取决于存储器映射实现的方式和操作系统。

存储器映射的一个强大功能是，每个逻辑存储器块可被赋予各种权限。例如，存储器可以是只读、只写、只能通过操作系统或给定的任务访问或在一组任务之间共享。通过确保物理存储器块只能被预先定义的任务访问，可以确保一个任务的执行不会导致另一个任务崩溃。有两种实现存储器管理的基本方式。使用固定大小的存储器块，称为页（page）；另一种使用可变大小的存储器块，称为段（segment）。

2. 地址翻译

存储器管理提供了两种不同的服务。第一种是将逻辑地址映射到可用的物理存储器中。

第二种功能发生在物理地址空间耗尽时（即由于数据不在随机访问存储器中，逻辑地址到物理地址的映射已不能完成）。

图 1-29 显示了一个页式存储系统是如何实现的。该例使用 24 位逻辑地址总线和 512KB 的存储器系统。来自处理器的 24 位逻辑地址被分成 16 位的偏移（被直接传给物理存储器）以及 8 位的页地址。页地址指向处理器当前访问的页（2^8=256 个页中的一个）。逻辑地址的位移量可以访问 64KB 页中的 2^{16} 个位置上的数据。

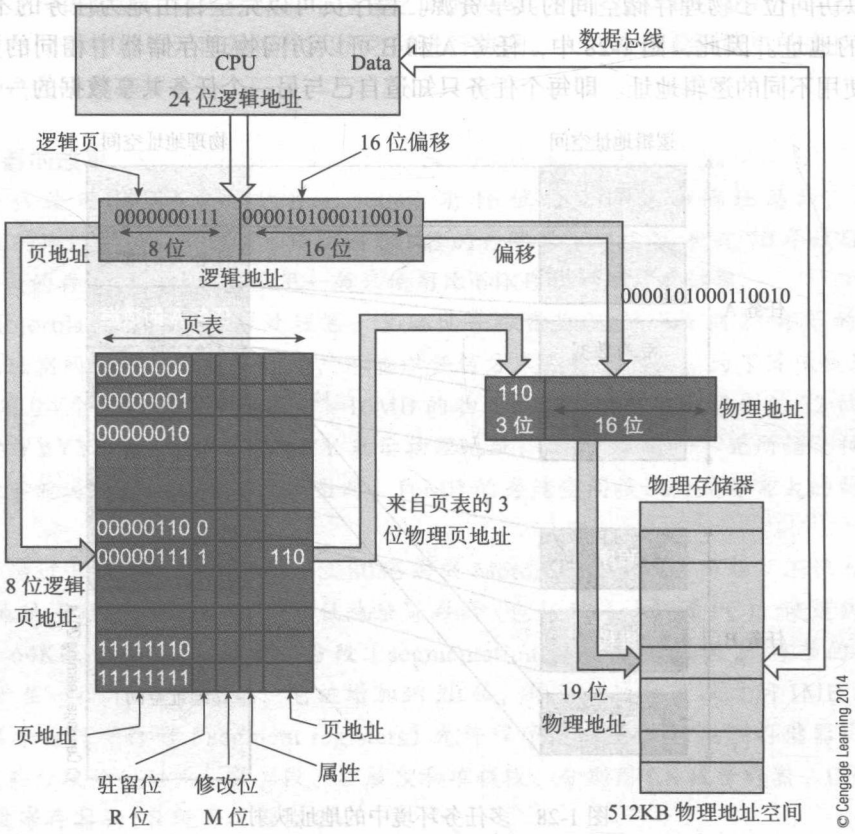


图 1-29 页表和虚拟存储器

页表包含 256 项，每一项对应一个逻辑页。例如，在图 1-29 中，CPU 正访问 8 位逻辑页地址 00000111_2。每项包含 3 位的页帧地址，是物理地址最重要的 3 位。该例中，物理页地址为 110。逻辑地址从 8+16 位变成了 3+16 位，逻辑地址 00000111 0000101000110010 被映射到物理地址 110 0000101000110010。

虽然在页表中有 256 个可能的项（每项对应一个逻辑页），物理页地址只有 3 位，它限制物理页只有 8 个。因此，随机访问存储器中不同的物理页不能与每个可能的逻辑页相对应。每个逻辑页地址有一位 R 字段与之相关联表示驻留。如果 R 位置为 1，表示该页当前在物理存储器中。如果 R 位被清零，对应的页将不在物理存储器中，页地址字段的内容将毫无意义。

每当产生逻辑地址且与当前逻辑页相关的 R 位被清除，将发生页故障（page fault）事件。一旦开始访问某个逻辑地址，由于 R 位被清除表示当前页不在存储器中，当前的指令必须暂停，因为它无法完成。

典型的微处理器具有总线错误输入引脚，表明存储器访问无法完成。当发生这种情况时，操作系统开始干预处理。虽然 CPU 试图访问的数据目前不在随机访问存储器中，但它在磁盘中。操作系统从磁盘检索包含所需存储器位置的页，将其加载到物理存储器中，并相应地更新页表。被暂停的指令可以继续执行。

该过程简单吗？完全不简单。当操作系统从磁盘获取一个新的页，它必须覆盖一个随机访问物理存储器的页。注意，虚拟存储器的一个功能是允许使用相对小的物理存储器来模拟大容量存储器。如果要用新页替换旧页，此时需要一个策略选择替换哪个旧页。经典的替换策略是最近最少使用（least recently used，LRU）算法，最长时间没有被访问的页将被新的页覆盖（也就是说，如果最近没有访问这个页，则在不久的将来也不太可能访问它）。

LRU 算法已在实践中被检验可以很好地工作。不幸的是，如果采用该策略，操作系统必须知道每个页是什么时候被访问的，这需要复杂的硬件（每个页必须在被访问后打上时间戳）。操作系统必须处理的另一个问题是，存储在 RAM 中和保存在磁盘上的数据之间的一致性问题。如果从磁盘上读取的页只包含程序的信息，它不会在 RAM 中被修改，因此，重写它不会导致任何问题。然而，如果页的内容是数据表或其他数据结构，它可能在 RAM 中被改写。在这种情况下，它不能被新的页覆盖。

在图 1-29 中，可以看到，页表中的每一项都有一个 M 位（修改位）。每当页由一个写操作访问，M 位被置位。当该页需要被覆盖，操作系统首先检查 M 位，如果被置位，则首先需要将被改写的页写回磁盘，然后再取新的页。

最后，当加载新的页后，将修改地址转换表，清除 M 位，将 R 位置位（表示该页是有效的），处理器可以重新执行被暂停的指令。

显然，每次页错误带来的开销非常大。只要页错误比较罕见，系统就可以因为访问局部性而工作正常。多数数据是簇聚在一起的，所以一旦从磁盘中载入一页，大部分要访问的数据都会在该页中找到。如果数据不是有序组织的或有许多不相关的任务，处理器将花费几乎所有的时间用于交换页，系统的有效工作将停顿。这种情况被称为抖动（thrashing）。抖动是指一种频繁访问资源导致计算机性能急剧下降的行为。然而，抖动在很大程度上是指由于需要加载和重新加载页而导致虚拟存储器系统发生故障的情况。

3. 两级表

图 1-29 给出的情况在现代高性能处理器中实际不存在。假设一个 32 位的计算机使用 13 位偏移（偏移指的是页内地址）访问 8KB 的页。这使得可以使用 32-13=19 位来选择 2^{19} 个逻辑页。在快速 RAM 中不可能构建这样大的页表（注意，相同问题在 Cache 的设计中也存在）。

图 1-30 给出了怎样使用多级（multi level）页表来实现地址转换而不需要大容量的页表。来自计算机的逻辑（虚拟）地址首先被分为 19 位的页地址和 13 位的页内偏移。页地址进一步被分为对应第一级页表的 10 位和对应第二级页表的 9 位。这两个表分别有 2^{10}=1024 项和 2^{9}=512 项。

图 1-30 是个简图，真正的页表包含更多地址转换过程的信息而不仅仅是指针。图 1-31 给出了 PowerPC 的地址转换表。

真正的页表结构包括多个指向其他表的指针。在这种层次地址转换表中，页表项包含指向下一级的指针。链表的终点为实际物理页，包含了 MMU 所需关于该页的信息。在实践中，典型的存储器管理单元包含的页表描述符可以描述以下信息。

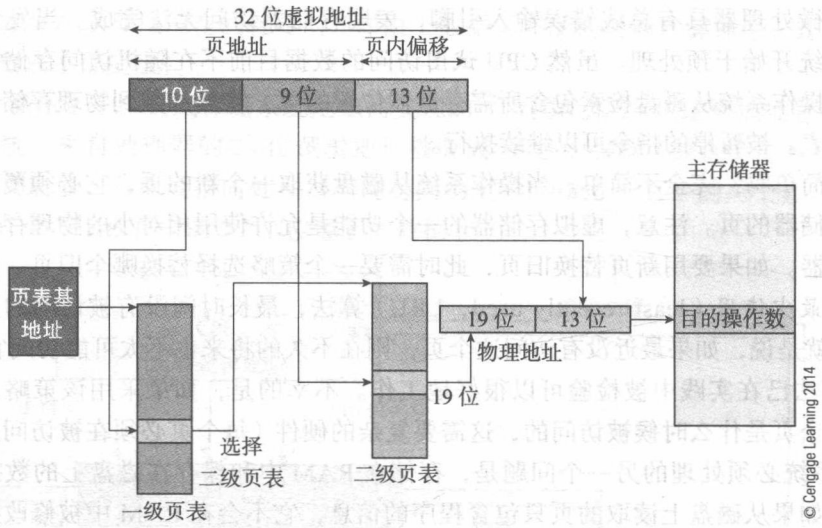


图 1-30 两级页表

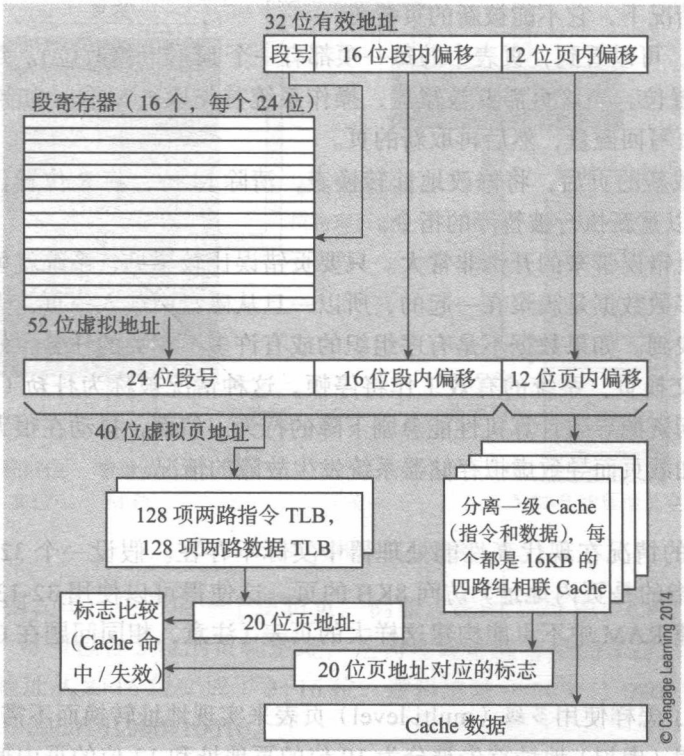


图 1-31 PowerPC 存储器管理系统

- **描述符类型**。描述符类型告诉 MMU 表是否存在下一级。
- **写保护位（W）**。表示其指向的页不可写。如果 W=1，树中的所有后续级别的写保护位都为 1。
- **使用位（U）**。该位在建立描述符表时被操作系统初始化为零。当描述符第一次被访问时，MMU 自动将其置为 1。虚拟存储器系统中，U 位用来描述当页需要被替换时，是否要将物理页写到磁盘。

- **管理员位（S）**。当该位被置位，所指向的页只能使用管理员方式访问（即操作系统级别的访问权限）。管理员态指的是操作系统运行的状态，具有比用户态更高的优先权。例如，如磁盘这样的 I/O 设备只能在管理员态下访问。
- **全局共享位（SG）**。设置为 1 时，表示页描述符可以共享，即如果 SG=1，系统中所有的任务都可以访问物理页。SG 告诉 MMU，在页表 Cache 中只要保持该页的一个描述符。*转换旁视缓冲器*（translation look-aside buffer，TLB）是一个小的全相联 Cache，可以通过同时搜索所有条目实现地址转换。
- **写访问级别（WAL）**。通过该描述符指出页的最低优先级。
- **读访问级别**。由 3 位组成，用于根据 WAL 位实现相应的读操作。
- **限制域（Limit）**。该域为转换表的下一级指出索引（index）值的下限或上限；即限制域限制了下一级表的大小。例如，逻辑地址域为 7 位，因此具有 128 项。然而，在实际系统中，在本级可能只有不超过 20 个页描述符。通过设置 Limit 为 5，可以限制表项为 32 个而不是 128 个。
- **上 / 下位（L/U）**。表示 Limit 位给出的是下限还是上限。如果 L/U=0，则 Limit 域包含索引的无符号上限，下一级表的所有表索引都必须小于或等于 Limit 域中的值。如果 L/U=1，则 Limit 域包含索引的无符号下限，所有表索引都必须大于或等于 Limit 域中的值。在这两种情况下，如果实际索引超出最大 / 最小值范围，将出现 Limit 违例。层次访问最终得到页描述符用来执行实际的逻辑地址到物理地址的转换。

逐级访问多级页表（例如，如图 1-30 所示）产生的页描述符将被用于实际的逻辑地址到物理地址的转换。除了上面给出的描述符外，页描述符可能还有以下的控制位：

- **修改位（M）**。指示相应的物理页是否被改写。由于 MMU 可以将其置位但不会清除，故该位在建立*描述符表*时被操作系统初始化为零。注意，使用位 U 在表描述符被访问后被置位，而 M 位在页被修改后置位。
- **锁位（L）**。表示相应的页描述符应规避 MMU 的页置换算法。当 L=1，物理页不能通过 MMU 替换。因此，可以使用 L 位来保持地址转换 Cache 中的页描述符。
- **Cache 抑制位（CI）**表明相应的页是否可以被缓存。如果 CI=1，那么该访问就不能被缓存。

本章小结

使用存储器可以将指令和数据保存在计算机中。由于技术本身的原因和受制造技术发展的限制，目前尚没有某种单一的设备或技术可以满足所有的个人计算机或工作站的需求。简单地说就是：速度很快的存储器价格昂贵，如果便宜的话速度就慢。

当开发大型计算机用于满足政府、工业以及军事需求时，设计人员很快发现，实际存储器系统的限制可以减少，或者说被屏蔽。的确，可以把不需要的数据存储在慢速的存储器中。用计算机的话说就是，把经常访问的数据保存在快速存储器中，而用慢速存储器实现数据归档。这种想法既简单又有效，对计算机性能产生了重要影响。但另一方面，它在实现起来可不简单。

计算机中有两种基本类型的存储器：主存储器中的随机访问半导体存储器（通常是 DRAM）和磁记录或光存储器等顺序访问的二级存储器。这两种存储器系统使用相似的技术来克服其速度限制：使用 Cache *存储器*（cache memory）来加速主存储器，使用*虚拟存储器*

（virtual memory）来加速二级存储器。Cache 存储器和虚拟存储器机制的原理相似，但在细节和实现技术上有很大不同。

本章第一部分介绍了用来加速主存储器的 Cache。相对小容量的 Cache（例如，4GB 的 DRAM 使用 1MB 的 Cache）可以显著提高系统的性能。因为数据和指令不是随机访问的，真实的数据和指令表现出时间（temporal）和空间（spatial）局部性，通常 CPU 需要访问存储器的数据 / 指令中超过 90% 的都可以在 Cache 中找到。如果找不到，必须从存储器中读取数据，并将数据副本放入 Cache。

本章研究了 Cache 的几个内容，如它是如何组织的；即物理存储器中的数据是如何被映射到容量较小的 Cache 中的。虽然直接映射 Cache 设计最简单，本章已经说明了它具有的局限性会降低其效率。本章还说明了如何利用它来构建组相联 Cache——这是一种在所有微处理器系统中都能找到的 Cache 类型。

本章的第二部分讨论虚拟存储器。硬盘读写速度比主存储器的 DRAM 慢若干数量级。虚拟存储器系统将存储器通常划分为 4KB 的页。4KB 的数据块可以从磁盘中调入主存储器中任意一个 4KB 的页中；即物理存储空间并不像处理器使用的虚拟或者逻辑地址空间那样是连续的。当计算机访问存储器，存储器管理单元将逻辑页地址转换为物理页地址，然后从存储器中的该页读取数据。然而，如果被访问的逻辑页不在存储器中，存储器管理单元就会产生页故障，操作系统开始从磁盘中将所需页调入主存储器中的页。当然，如果所有的物理页当前都在使用，操作系统必须牺牲一个页，替换它，再装入新的页。

习题

1.1 Cache 是什么？为什么计算机要使用 Cache？

1.2 解释下列术语的含义？时间局部性；空间的局部性。

1.3 试推导出具有 Cache 的存储器系统的加速比表达式（假定命中率为 h，主存储器访问时间是 Cache 访问时间的 k 倍，其中 $k>1$）。假设该系统是一个理想的系统，不必考虑时钟周期时间的影响。

1.4 对以下的理想系统，计算加速比 S。每种情况下，t_c 为 Cache 的访问时间，t_m 是主存储器的访问时间，h 为命中率。

a. $t_m = 60$ ns, $t_c = 3$ ns, $h = 0.99$

b. $t_m = 60$ ns, $t_c = 3$ ns, $h = 0.90$

c. $t_m = 60$ ns, $t_c = 3$ ns, $h = 0.80$

d. $t_m = 60$ ns, $t_c = 3$ ns, $h = 0.70$

1.5 对以下的理想系统，计算得到给定加速比 S 所需的命中率 h。

a. $t_m = 50$ ns, $t_c = 2$ ns, $S = 1.5$

b. $t_m = 50$ ns, $t_c = 2$ ns, $S = 5.0$

c. $t_m = 50$ ns, $t_c = 2$ ns, $S = 10.0$

d. $t_m = 50$ ns, $t_c = 2$ ns, $S = 20.0$

1.6 微处理器的操作通常在给定时间间隔内完成（即时钟周期的整数倍）。例如，如果时钟为 100MHz，时钟周期时间就是 10ns，所有操作的时间必须为 10ns 的整数倍。下面的数据中 t_{cyc} 为处理器的时钟周期时间，t_m 为主存储器的访问时间（包括所有开销，例如地址译码）。针对每种情况，计算访问存储器实际需要的时间。

a. $t_{cyc} = 50$ ns, $t_m = 100$ ns

b. $t_{cyc} = 50$ ns, $t_m = 105$ ns

c. t_{cyc} = 10 ns, t_m = 75 ns

d. t_{cyc} = 20 ns, t_m = 75 ns

1.7 针对以下微处理器系统，计算当 h 接近 100% 时可以获得的最大加速比。

a. t_{cyc} = 20 ns, t_m = 75 ns, t_c = 25 ns

b. t_{cyc} = 10 ns, t_m = 75 ns, t_c = 15 ns

c. t_{cyc} = 20 ns, t_m = 75 ns, t_c = 15 ns

1.8 实际使用中，计算机有时在执行存储器访问的时候执行内部操作。因此，降低了有效加速比，因为 Cache 对内部操作没有作用。执行一条指令所需的平均时间可以写为：

$$t_{ave} = F_{internal} \cdot t_{cyc} + F_{memory}[ht_c + (1-h)(t_c + t_d)] \cdot t_{cyc}$$

其中：

$F_{internal}$ 为执行内部操作所花时间所占的比例（0～1 之间）；

F_{memory} 为执行存储器访问操作所花时间所占的比例（0～1 之间）；

t_{cyc} 为时钟周期时间；

t_c 为以时间周期为单位的 Cache 访问时间；

t_d 为以时间周期为单位的 Cache 失效开销；

请计算下列系统的平均周期时间。

a. $F_{internal}$ = 20%, t_{cyc} = 20 ns, t_c = 1, t_d = 3, h = 0.95

b. $F_{internal}$ = 50%, t_{cyc} = 20 ns, t_c = 1, t_d = 3, h = 0.9

1.9 针对上题中的系统，如果参数变成：$F_{internal}$ = 40%, t_{cyc} = 20 ns, t_c = 1, t_d = 4，试计算将平均指令时间减半所需的命中率。

1.10 主存储器的数据是如何映射到如下 Cache 中的？

a. 直接映射 Cache

b. 全相联 Cache

c. 组相联 Cache

1.11 在一个直接映射 Cache 存储器系统中，下列术语的含义是什么？

a. 字

b. 块

c. 组

1.12 为什么构建一个全相联 Cache 很困难？为什么组相联 Cache 如此受欢迎？

1.13 画图说明如何使用 Cache 标志 RAM 来实现直接映射 Cache。对比直接映射 Cache 和全相联 Cache 的优缺点。

1.14 什么是 Cache 一致性？

1.15 突发模式的操作是什么（在具有 Cache 的环境下）？

1.16 按道理，Cache 是非常简单的概念，只需要将经常访问的数据放在高速 RAM 中即可。在实践中，相对其他计算机部件，Cache 存储器系统的设计是很困难的。这种说法是否正确？

1.17 讨论工程师在为 Cache 选择合适的块大小（容量）时需要考虑的因素。

1.18 Cache 系统可以位于 CPU 和 MMU 之间（即逻辑 Cache）或在 MMU 与系统随机访问存储器（即物理 Cache）之间。是什么因素决定了 Cache 的最佳位置？

1.19 当 CPU 写 Cache 时，Cache 中的项和存储器中对应的项都需要更新。如果数据不在 Cache 中，它必须从存储器中取出，然后装入 Cache。如果 t_1 为 Cache 失效时重新加载需要的时间，证明存储器系统的有效平均访问时间由下式给出：

$$t_{ave} = ht_c + (1-h)t_m + (1-h)t_1$$

1.20 为什么设计数据 Cache 比设计指令 Cache 要困难？

1.21 Cache 可以相对于主存储器以串行或并行的方式工作。在串行访问模式下，在 Cache 中查找数据，如果发生失效，然后再访问主存。在并行访问模式下，Cache 和主存储器同时被访问。如果命中，中止对主存储器的访问。假设系统的命中率为 h，Cache 访问时间与主存储器访问时间的比例为 k（$k<1$）。推导出的并行访问 Cache 和串行访问 Cache 加速比的表达式。

1.22 如果使用串行访问 Cache，可以容忍其加速比比并行访问 Cache 的加速比小 5%，此时命中率为多少才能达到要求？假定主存储器的访问时间为 30ns，Cache 的访问时间为 3ns。

1.23 什么是一级 Cache 和二级 Cache（即 L1 和 L2 Cache）？

1.24 某系统具有一级 Cache 和二级 Cache。一级 Cache 的命中率为 90%，二级 Cache 的命中率为 80%。访问一级 Cache 需要 1 个周期，访问二级 Cache 需要 4 个周期，访问主存储器需要 50 个周期。平均访问时间是多少？

1.25 计算机的 Cache 访问时间为一个周期，平均命中率为 95%，失效开销为 100 个周期。

a. 该计算机的平均周期时间是多少？

b. 如果增加二级 Cache 的命中率为 80%，其失效开销为 6 个周期。对平均周期时间的影响如何？

c. 具有一级 Cache 和二级 Cache 的计算机执行了 10000 次存储器访问。测试程序运行时，记录到一级 Cache 失效了 500 次，二级 Cache 失效了 300 次。一级 Cache 和二级 Cache 的总体失效率是多少？

1.26 考虑多级 Cache 命中率，局部命中率和总体命中率之间有何区别？

1.27 为什么经常引用（使用）失效率而不是命中率？什么是 Victim Cache，如何使用它？Victim Cache 能够减少何种类型的失效？Victim Cache 和 Annex Cache 的本质区别是什么？

1.28 某处理器的存储器管理使用 4KB 大小的页。它有一个 32KB 的 Cache，Cache 块大小为 16B。为了加快存储器访问，可以使访问 Cache 与逻辑到物理地址的转换同时进行。为了实现该方案，需要实现什么样的相联度？

1.29 假设混合 Cache 具有以下特性：

Cache 读写开销	1 个周期
失效率	3%
Load 指令所占比例	20%
Store 指令所占比例	5%
失效开销	20 个周期

平均访问时间是多少？

1.30 某 64 位处理器有 8MB 的四路组相联 Cache，块大小为 32B。地址位是如何按照组、块、块内偏移位进行设置的？

1.31 某计算机具有分离的数据 Cache，采用写回策略。Cache 块大小为 64B。读访问占全部存储器操作的 80%。处理器、存储器和数据总线都是 64 位的。主存储器访问延迟（第一次访问）为 20 个周期，后续访问需要 2 个周期。Cache 命中率为 96%。试计算 Cache 的失效开销。

1.32 导致 Cache 失效的 3 个原因是：强制失效、容量失效和冲突失效。给出这些术语的定义。简要解释如何尽量减少它们的影响。

1.33 CPU 中的 Cache 和硬盘中的 Cache 有什么根本差异？

1.34 为什么在使用硬盘的系统中必须使用存储器管理？存储器管理可以提供哪些保护功能？存储器管理系统具有保护功能，该功能在 Cache 中也存在吗？

1.35 写回 Cache 和写直达 Cache 之间有何差异，对系统性能有何影响？

1.36 32 位地址体系结构计算机的存储器管理系统具有一级 4KB 的页表。该页表对应多大的存储器空间？

1.37 考虑全相联的 16 字节的 Cache 具有 4 个块，每块 4 个字。Cache 使用 LRU（最近最少使用）算法处理块替换。Cache 初始是空的，块从第 0 块开始装载。

给定下面的十六进制地址序列，指出命中或失效情况。给出所有读操作结束后 Cache 的状态。

00 03 05 08 13 14 11 04 0F 0C 23 00 01 02 04 06 05 07 09 21

1.38 计算机指令集具有下表中的特点。

类别	指令频度	周期
运算操作	70%	1
条件操作	15%	2
Load 操作	10%	2
Store 操作	5%	2
命中率	95%	
Cache 读失效开销 10 个周期		
写直达时间 5 个周期（写入存储器，不缓存）		

每条指令的平均周期数是多少？

1.39 考虑下面的代码，访问存储器中的整数 x 和 s 以及整数向量 $y[i]$ 这 3 个值。

```
for (j = 0; j < 100; j++)
{
x = y[j];
s = s + x;
}
```

该系统具有一级 Cache 和二级 Cache。一级 Cache 的访问时间是 2 个周期，二级 Cache 的访问时间是 6 个周期，主存储器的访问时间为 50 个周期。在此情况下，所有的存储器和 Cache 的访问是并行执行的。假设没有缓存数组，每次新的对数组的访问都将导致失效。每个循环中以时钟周期数表示的存储器延迟（第一次迭代后）为多少？

1.40 假设采用与题 1.39 相同的系统，如果采用预取技术，每次对主存储器的访问将导致下一块调入二级 Cache，此时循环的平均存储器延迟为多少？假设对二级 Cache 采用预取技术不会导致进一步的存储器访问开销。

1.41 某 64 位计算机有 128KB 的八路组相联 Cache。Cache 有 128 组，每块为 16 个字。每个地址需要多少位标志位？

1.42 哪种类型的 Cache，可以用来减少由于频繁交换而导致的 Cache 抖动？

1.43 给定以下数据，假设时钟频率为 1000MHz。

存储器	命中时间	失效率
一级 Cache	1 个周期	2%
二级 Cache	8 个周期	5%
DRAM	20 个周期	0.1%
磁盘	10ms	

计算平均存储器访问时间。假定二级 Cache 和 DRAM 可以与一级 Cache 并行访问。

1.44 某 16 位 CPU 的 Cache 有 32 块，每块 16B。CPU 访问一个字节的十进制地址为 3210。这将导致失效，调入新块。该块位于 Cache 中的什么位置？

1.45 某计算机具有 256B 的地址空间和两路 32B 的组相联 Cache。计算机字的大小是一个字节，每个 Cache 块包含 4 个字节，每个 Cache 有 4 块。如果 Cache 最初是空的，按照如下十六进制地址序列读取，显示相应的命中和失效的情况。

48, 0C, 48, 4C, 5C, 3A, 20, 21, 22, 24, 81, 49, 30, 34, 27, 3E, 24, 28, 2C, 40

1.46 人们总是在寻找更有效的 Cache 机制，特别是减少失效开销的方法（例如，使用 Annex Cache 或 Victim Cache）。某个学生提出下面的建议。并非所有数据都相同。有些数据使用比别的更频繁，特别是数值小的数。所以为什么不这样安排 Cache：当有两个候选位置的块可能被替换时，首先

把数值大的块替换出去？这种方法是否可行？

1.47 计算机 Cache 的命中率为 95%，每块为 4 个 32 位的字。处理器平均访问 Cache 的频率为 1 亿次 /s。20% 的 CPU 操作为 load/store 操作（其中读和写操作分别占 70% 和 30%）。Cache 使用写直达机制，在写失效时会替换一个块。在此情况下，32 位 CPU- 存储器总线可使用的带宽是多少？

1.48 重做题 1.47。此时 Cache 使用写回方式，Cache 中平均 25% 的块为脏块（已被修改）。

1.49 计算机的存储器容量为 256 个字，Cache 容量为 16 个字。按照以下地址顺序读数据：

0, 1, 2, 3, 4, 5, 10, 13, 16, 19, 21, 4, 8, 12, 30, 40, 41, 42, 35, 1, 3, 13

假设所有 Cache 块最初都是无效的，指出 Cache 是如何访问的。针对以下几种 Cache，为每次访问做出标记：命中、容量失效、强制失效或者冲突失效。

a. 全相联 Cache

b. 两路组相联 Cache

c. 直接映射 Cache

1.50 某系统的存储器访问时间为 50ns，Cache 访问时间为 2ns。指令执行时间为 4ns（不包括存储器访问），平均每条指令需要 0.25 次存储器访问。如果命中率是 0.90，指令的平均时间是多少？

1.51 某计算机具有 24 位地址总线，存储器容量为 16MB，Cache 容量为 64KB。字长为两个字节。

a. 块容量为 32 个字的直接映射 Cache 的地址格式是什么？

b. 块容量为 32 个字的全相联 Cache 的地址格式是什么？

c. 块容量为 32 个字的四路组相联 Cache 的地址格式是什么？

1.52 某计算机的存储器访问时间为 38ns 且不使用 Cache。为其添加一个访问时间为 10ns 的 Cache。计算机的运行速度提高了 90%。试估计其命中率是多少。

1.53 为什么二级 Cache 的命中率通常低于一级 Cache 的命中率？

1.54 某系统一级 Cache 的命中率为 87%（命中需要 1 个周期）。二级 Cache 的命中率为 90%，访问时间为 10 个周期。主存储器的访问时间为 200 个周期。平均访问时间是多少？

第 2 章

Computer Organization and Architecture: Themes and Variations

主存储器

“记忆才是我们真正拥有的。”

——Elias Lieberman

“逝者活在人们的记忆里。”

——Cicero

“虽然太阳系以每小时四万三千英里[⊖]的速度向武仙座球状星团 M13 靠近，但仍有些人坚持不承认这个过程。”

——Kurt Vonnegut，泰坦的警报

“计算机应该由玻璃构成，因为它到处都是瓶颈。”

——Alan Clements，2001.03

“在这个世界上，我们身处何地并不重要，我们朝着什么方向前行才重要。”

——Oliver Wendell Holmes

“瞄准是不够的，你必须命中。”

——德国谚语

“……计算机硬件发展十分迅速。人类有史以来没有其他技术能够在 30 年中获得 6 个数量级的性能－价格收益。”

——Fred Brooks Jr

2.1 简介

本章将介绍的计算机存储器和 I/O 系统是其主存储器（main memory）和直接访问存储器（immediate access store）。从许多方面看，这是计算机最乏味的部分，其作用就是负责为处理器芯片存储数据。存储器似乎没有什么可以炫耀的，从来没有听到电脑爱好者像谈论超频或多处理器那样兴奋地谈论它。简而言之，主存储器保存了一些数据直到 Cache 需要这些数据或者关闭了计算机电源。但是，本章将要介绍一些多年来在存储技术上取得的显著进步。将研究那些出现和已克服的问题，并介绍一些正在研发的新技术和物理器件。

在这一章中，将介绍直接访问存储器的工作原理，静态（static）和动态（dynamic）存储器之间的差异，易失性（volatile）和非易失性（nonvolatile）存储器之间的差异。此外还介绍一些开始在存储系统中发挥重要作用的新技术。本章还将讨论包括存储设备特点在内的存储系统设计者必须考虑的问题。本书的大多数读者都不可能设计一个 CPU，但可能会为嵌

⊖ 1 英里 =1609.344 米。——编辑注

入式计算机或类似装置设计一个存储系统。从计算机摄像头到手机这样的数字系统，其存储系统往往由现成的组件构成。

动态——用词不当

大多数计算机的主存储器使用动态存储器（DRAM）。动态（dynamic）这个词有误导作用。英语中动态一词通常表示性能提升是正面的，甚至是积极的。然而，在DRAM中使用其含义正好相反。数据是以电荷的形式存储的，在几毫秒内就会泄漏光。换句话说，DRAM在几毫秒内就会失去其保存的数据。要想在DRAM中保存数据，只有在其消失之前不断地读取它然后再把它写回。动态这个术语表示了这种属性。（也许DRAM中的“D”应该表示为滴漏（drippy）。）

原理上，计算机存储器是最容易理解的部件；它是保存程序和数据的地方。在实践中，存储系统总由从DRAM到硬盘这样的各种设备构成，这些设备又是由多种技术制造的。事实上，存储设备性能的差异比计算机系统中的其他部件都要大。存储部件可能保持几个字或几百GB的数据；其读取时间最短为1ns最长为几秒；其价格可能从几美分到一千美元。

因为存储器涉及的内容很多，涵盖不同的存储技术，本书将存储部分分为两章。本章着眼于程序运行时所在的直接访问存储器（immediate access store）或主存储器（primary storage）。下一章将介绍用来保存当前没有执行的程序和数据的二级存储（secondary storage）系统。虽然现在有的二级存储器采用了固态硬盘，一般情况下，主存储器采用半导体技术、而二级存储采用磁性或光学技术实现。

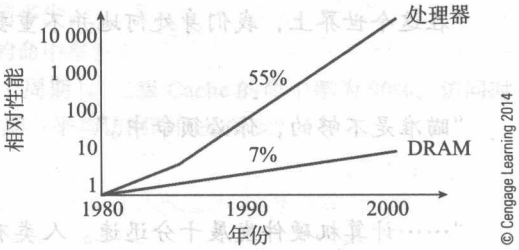


图2-1 20年来存储器技术的发展

存储器的性能提升迅速，在过去的20年中每年约提升7%，如图2-1所示。另一方面，处理器的性能提升更了不起，每年提高60%，比存储器性能提升快很多，这使得存储器系统成为现代处理器的一大瓶颈。

本章将介绍静态和动态半导体存储器的工作原理，以及它们是如何与处理器连接的。特别是将介绍存储器的时序，给出数据传送过程中操作的序列。这部分内容将帮助读者理解在第4章中将介绍的一些概念，第4章主要介绍输入/输出技术。本章有一节是专门介绍DRAM的，它具有复杂的接口要求，近年来已经出现了几种形式的DRAM。

2.1.1 存储系统的原理和参数

人们期望存储设备能够完成其预期工作——记住数据。可以利用任何一种材料来实现存储器，只要该材料的某种性质可以改变，而后又可以检测这种变化即可。大多数计算机教材认为，第一个存储系统是Hollerith发明的制表机上的打孔卡片或者是Jacquard织布机上的木制卡。当然，从广义上讲，所有包括从埃及象形文字到纸笔系统等在内的各种书写系统都是存储系统。

可以用来存储数据的物理性质的范围相当广泛。虽然人们都听到过山谷的回声，但很少有人会想到使用回声作为存储系统的基础。20世纪40年代后期，人们利用声波在水银管内的传播来存储数据（水银延迟线存储器）。在该管的一端，以一系列超声波脉冲的形式将数

据传输到管中（声音在水银中的传输速度为1450m/s）。当声音到达管子的另一端，它由声电转换器接收并放大，并反馈到管子的前一端。这是一种动态存储器，因为数据总是以物理运动的形式存在。即使在今天，玻璃延迟线仍然用在一些电视信号处理中，用来将信号延迟一段固定的时间。

20世纪50年代至70年代，数据被存储在由铁氧体磁芯（ferrite core）构成的小珠子（或环）的磁场中（因此计算机文献中通常将其称为磁芯存储器）。今天的硬盘仍然使用相同的磁性质来存储数据。正是由于其重要性，本书将在下一章详细讨论磁记录方式。

从20世纪70年代中期开始，半导体存储器已经成为主存储器的标准形式，即静态RAM（SRAM）或动态RAM（DRAM）。今天只有小型嵌入式系统使用静态RAM，而PC使用大约2～48GB的DRAM。如果DRAM是本章的主题（theme）的话，那么变化（variation）则是由一些半导体存储器的新形式，如铁电（ferroelectric）半导体存储器，它用晶体中原子的位置来存储数据；或者双向开关（ovonic）存储器，它通过硫系（chalcogenide）玻璃在非晶态（amorphous）和晶态（polycrystalline）之间的转换来存储数据。下面引入一些词汇来对存储器系统和技术进行描述。

1. 随机访问和顺序访问存储器

计算机所使用的不同存储技术之间的根本区别在于数据的访问方式：直接（directly）访问或顺序（sequentially）访问。可以直接访问的存储器被称为随机访问存储器（random access memory，RAM），这是由于可以随机地访问任何数据元素，且所花费的时间是固定的，与数据的物理位置无关。这种存储器也被称为直接访问存储器（immediate access memory，IAS）。当然，这些存储器肯定不是立即访问的——没有哪种存储器可以立即得到访问结果，只是它们比其他类型存储器的速度要快罢了。

顺序访问存储器需要依次访问每个存储单元直到找到所需的元素。顺序访问存储器的例子是磁带存储器，需要一直读磁带直到找到所要的数据。前面提到的水银延迟线（acoustic mercury delay line）存储器就是顺序访问存储器。随机访问存储器比顺序访问存储器要快，但它们也更贵。大多数半导体存储器如DRAM或闪存都是随机访问的。而移位寄存器是一种顺序访问存储器。

随机访问和顺序访问

需要注意的是，硬件和软件开发者有时使用不同的方式进行串行访问。如果获取所需数据需要经过搜索若干其他元素才能完成，则存储器访问方式就是顺序的。在软件世界中，如果访问磁盘中某个文件而不需要访问磁盘中的其他文件，则认为其为随机访问。然而，底层的存储设备，例如硬盘，是顺序访问的，这是因为必须等待所需的数据旋转到磁头下方。

人们常说存储器的速度或者说其是如何快或如何慢。这些术语指的是需要多长的时间来访问数据。存储器的关键参数为访问时间 t_{acc}。

2. 易失性和非易失性存储器

在理想的存储器中，数据将一直保持不变直到修改它。这种存储器被称为非易失性（nonvolatile）存储器。例如，写数据到硬盘，这些数据将一直保存在硬盘上。有些存储器技术只在加电的时候才能保留数据，拔掉电源插头数据就丢失了。这些存储器被称为易失性（volatile）存储器，因为断电后数据将丢失。大多数PC和工作站的主存都由易失性的

DRAM 构成。如果采用非易失性存储器，就没有必要在每次打开电源开关后重新启动计算机（即将操作系统从非易失性的磁盘存储器装载到计算机的易失性存储器中）。稍后将介绍一类非易失性存储器——闪存（flash memory），它是目前的一种主流技术，是非易失性存储器的新形式。

3. 读 / 写和只读存储器

如果可以向存储器写或者从存储器读数据，而且可以完成的读操作与写操作的次数相近，这种存储器就是读 / 写（read/write）存储器。计算机的主存当然是由读 / 写存储器组成。如果可以很容易地读存储器，但其内容不能修改，则为只读（read-only）存储器。只读存储器总是非易失性存储器[⊖]。

当然，没有真正的只读存储器。如果有，就不可能首先把数据放入其中。完全意义上的只读存储器正如王水的地位一样，除非被分解，否则没有任何容器能够容纳它。在掩模编程（mask-programmed）ROM 被生产的时候数据就完成了加载，因为每个存储单元的物理结构决定了它保存的是 1 还是 0。掩模编程 ROM 价格便宜，但以后不能修改。它曾经用来保存引导程序、操作系统以及 BIOS。今天它已过时，被闪存替代了。

实际的只读存储器更应该描述为大部时间为读（read-mostly）的存储器，它可以进行有限次数的修改。此外，它的写操作要比读操作更复杂且更缓慢。这类存储器的例子包括 EPROM、EEPROM 和闪存。EPROM 表示电可编程只读存储器（electronically programmable read-only memory），EEPROM 表示可擦除和电可编程只读存储器（erasable and electronically programmable read-only memory）。后文将更详细地介绍这些技术。

4. 静态和动态存储器

随机访问、读写以及易失性存储器可以分为两类：静态（static）和动态（dynamic）。它是通过半导体存储单元的结构及其性质来进行分类的。静态存储器利用交叉耦合的晶体管来构造一个 RS 触发器，用触发器的状态来存储数据。动态存储器（DRAM）采用半导体技术将数据表示为存储在电容中的电荷。相比于动态存储器，静态存储器的速度较快、价格较昂贵、芯片密度（位 / 芯片）较低。动态存储器更便宜，但相比静态存储器在实际电路中使用更难。今天这种情况在逐渐好转。如今，通过 CPU、主板桥芯片和 DRAM 自己来完成对 DRAM 的控制。20 世纪 80 年代，在计算机中实现 DRAM 控制器是一项艰巨的任务。

引脚

存储部件的接口是引脚（连接线）。存储器到底需要多少连接线？关注这个问题的原因是它决定了印刷电路板或主板的复杂性。

考虑一个静态 RAM，由 64K 个 8 位的字组成。该设备需要 16 根地址线来访问 64K 个字和 8 根数据线。需要 8 位数据线，是因为它访问的是字节。最后，它需要两根电源线和读 / 写引脚来选择存储器读或写以及 CS(片选) 信号线使芯片可以参与数据传输（因为它可能只是一个存储阵列中成百上千芯片中的一员）。

因此，最小连接线数量是 16+8+2+2=28。

动态存储单元中数据将在几毫秒后丢失，除非它通过不断的刷新（refreshing）操作重

⊖ 我想说的是，易失性只读存储器是一个矛盾体。然而，在我看来，易失性 ROM 至少有一个重要的应用。读者你能想到吗？

写。动态存储器中读与写的访问时间不同，典型的 DRAM 并不能真正地随机访问，这是因为相邻存储单元要比随机选择单元的访问速度更快。

由于 DRAM 组成了大多数 PC 和工作站的存储器，DRAM 的性能和特点对计算机的整体性能有重要影响。后文将详细介绍 DRAM。

5. 存储器参数

存储器的最小单元就是存储一位的*存储位元*（memory cell）。半导体存储器被组织为一个 n 行 m 列的数组；即它含有 $n \times m$ 个位元。存储器的宽度（width）m，就是存储器中每个字包含的比特（位）数。当执行读或写操作时，一个字的所有 m 位同时进行相同的操作。存储器的长度（length）n，定义为地址的数目（即存储器使用 $\log_2 n$ 位地址线来访问 n 个位置。

存储部件的宽度并不一定与计算机中总线或基本数据单元的宽度相同。例如，计算机可能具有 64 位的数据总线，但使用 4 位宽度的存储部件。因为每个存储部件只提供 4 位数据，这就需要 64/4=16 个存储部件并排工作以满足 64 位数据总线的需求。如果每个 4 位存储部件具有 4M 个可寻址位置，则存储部件的容量为 $4b \times 4M=16Mb=2^{24}b=2MB$，整个存储器的容量为 16 片 $\times 16Mb=2^{28}b=32MB$。

图 2-2 说明了 3 种宽度之间的关系。该 CPU 具有 64 位的寄存器，因此被称为 64 位机；即它是 64 位体系结构的。CPU 和存储器之间的总线为 32 位宽，64 位的数据元素必须连续两次通过总线操作从存储器中取出。此时可以说，64 位*体系结构*（architecture）用 32 位*组织形式*（organization）实现。该存储器阵列由 4 个 8 位芯片构成。这些芯片在每个读周期贡献 8 位数据，即芯片是并行访问的。

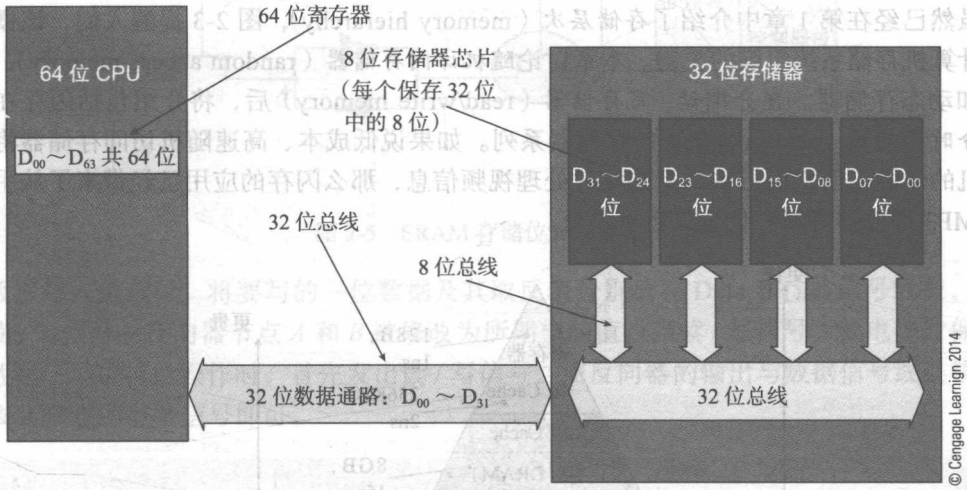


图 2-2 存储器芯片宽度、存储器宽度和总线宽度

存储器的主要时间参数是*其读访问时间*（read access time）、*写访问时间*（write access time）和*周期时间*（cycle time）。读访问时间就是访问存储器位置并获取其内容的时间。写访问时间就是将数据写到存储设备中的时间。周期时间是两个连续的存储器访问之间的最小间隔时间。

理想情况下，存储器的读、写和周期时间都应该相同，这在半导体静态 RAM 中是正确的。有些存储器（例如，DRAM）的周期时间比读或写访问时间要长，这是因为在连续访

问之间需要完成某些内部操作。前文已经指出，大部时间为读的存储器，如 Flash EPROM，写访问时间要比读访问时间长得多。

一个特别重要的存储器参数是*功耗*（power consumption）。半导体读/写存储器需要能量来操作和存储数据。功耗对于由电池供电的便携式设备设计来说是十分重要的；这是当今泛在计算和个人计算关注的一个重要问题。

存储设备（或任意其他部件）消耗的能量最终表现为热量，必须被扩散到周围的环境中。由于散热量决定了存储器（或 CPU）的操作温度，因此散热非常重要。当某个设备对象的温度高于环境温度，散热量与该设备的散热面积、散热表面的效率、环境温度和设备温度之间的差值等有关。某设备的散热量 $P_{dissipate}$ 由下式给出：

$$P_{dissipate}=K\cdot A\cdot(t_{device}-t_{ambient})$$

其中：K 为常数，A 为散热面面积。

因为部件失效的概率是温度的指数函数（每增加 15℃，故障率增加一倍），因此功耗应尽可能地小。具有较大散热量的芯片，如处理器，需要配备*散热片*（heat sink）来增加其表面积；如果散热面积 A 增加了，散热需要的温度差（$t_{device}-t_{ambient}$）就可以减小。

表 2-1 总结了本章中讨论的几种存储设备的典型特性。

表 2-1 半导体存储器的特性

特性	DRAM	SRAM	闪存
静态	否	是	是
易失性	是	是	否
典型容量	256Mb	64Mb	256Mb
组织	4b × 64M	8b × 8M	8b × 32M
访问时间	25ns	2ns	40ns
应用场合	主存储器	Cache 存储器	BIOS，数字播放器，MP3

2.1.2 存储层次

虽然已经在第 1 章中介绍了*存储层次*（memory hierarchy），图 2-3 提醒人们，该概念是理解计算机存储系统组织的关键。本章讨论*随机访问存储器*（random access memory），包括静态和动态存储器。在介绍*读/写存储器*（read/write memory）后，将介绍包括闪存在内的*大部分时间为读*（read-mostly）的存储器系列。如果说低成本、高速随机访问存储器将改进计算机的性能，使它们更快、更能实时处理视频信息，那么闪存的应用已经带来了从手持设备到 MP3 播放器等新一代个人数字系统。

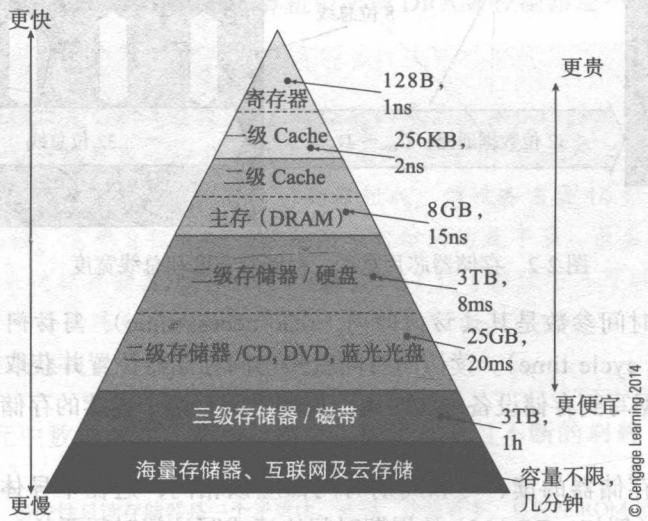


图 2-3 存储层次

2.2 主存储器

这一节将介绍直接访问存储器（immediate access store）或者主存储器（primary memory）。没有低成本、高速的存储器，实现复杂的操作系统和应用（例如 Photoshop）是十分困难的。本节首先介绍静态存储器（SRAM），因为它在动态存储器（DRAM）之前出现且比 DRAM 更容易理解。实际上，Cache 存储器通常由 SRAM 实现。

2.2.1 SRAM

图 2-4 说明了 SRAM 概念上是如何工作的。两个反向器端端相联构成一个环。门 1 的输入是 A，其输出 $B=\overline{A}$，这又是门 2 的输入。门 2 的输出是 A，其中 $A=\overline{B}=\overline{\overline{A}}=A$，这也是门 1 的输入。门 1 的输入是由门 1 的输入经过反馈产生的。这是一个自主维持存储器（self-sustaining memory）。无论初始给门 1 何种输入，其反馈都会使这种状态得到维持。

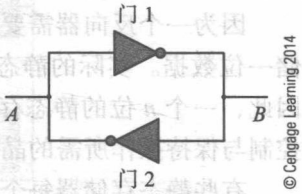


图 2-4 SRAM 原理

图 2-5 显示了如何将图 2-4 中的两个交叉耦合的反向器变成实际的静态存储位元。作为开关的两个晶体管[⊖]连接到反向器电路的左边和右边，用于访问存储单元（右边的阴影插图显示了晶体管有 3 个端子；在控制栅极的信号电平决定了另外两极之间的路径是打开还是关闭）。当读 / 写信号线是低电平，这两个晶体管都处于打开状态，数据信号线 Data 和 $\overline{\text{Data}}$ 都没有被反向器驱动（即被控制）。

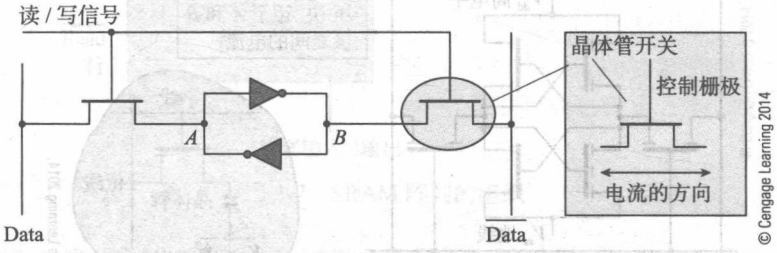


图 2-5 SRAM 存储位元的操作

数据写入位元时，将要写的一位数据及其取反值分别放在 Data 和 $\overline{\text{Data}}$ 信号线上，然后发出读 / 写信号。反向器节点 A 和 B 被修改为所期望的值，当读 / 写信号为低电平时保持写入的数据。当进行读操作时，首先发出读 / 写信号，使反向器的输出与数据信号线连接，然后读取 A 节点的电平信号即可。

SRAM

当前 SRAM 在许多方面都没有以前那么重要了。由于其与 CPU 的接口比 DRAM 的要简单，它曾经是许多（特别是小型）计算机系统存储器的主要选择。此外，SRAM 可以使用一小块电池就可以在掉电模式下保持数据，这是 SRAM 在闪存被广泛使用前具有重要价值的原因。今天，由于 DRAM 存储器的经济性，SRAM 对大型系统设计师来说

⊖ 这里没有详细解释晶体管的操作。只要将其想象为图 2-5 中插图所示的那样一个具有 3 个端子（连接线）的装置即可。其中两端通过开关连接；即，两端间可以存在也可以不存在电路。第三端为控制端，用来改变开关的状态。若端子为 A、B 和 C（其中 C 为控制端），如果 C=1，则 A 与 B 相联；否则 A 与 B 断开。

并不重要了。

这里讨论SRAM是因为它（例如，对于小型或低功耗系统来说）仍然是重要的，且对其时序特性的介绍可以作为介绍DRAM更复杂行为的基础。此外，SRAM与微处理器的接口与其他存储器映射外设（将在第4章中讨论）有很大不同。

然而，由于静态存储器非常快，它仍然可以用于制造Cache存储器。早在1981年，实验室中就观测到了低于0.6ns的SRAM访问时间（当时使用的原材料是砷化镓而非硅）。

因为一个反向器需要由两个串联的晶体管来实现，所以每个静态位元需要6个晶体管存储一位数据。实际的静态存储器为$m \times n$的位元阵列。访问位元时需要指定其行和列地址。因此，一个n位的静态存储单元需要$6n$个晶体管加上用于行列地址解码以及执行各种信号控制与保持操作所需的晶体管。

有些静态存储器每个位元只需使用4个晶体管。无论怎样，n位静态存储器位元需要$4n$或$6n$个晶体管。而动态存储位元只需要一个晶体管，这意味着动态存储器的密度至少可以达到静态存储器的4倍。图2-6给出了由6个晶体管构成的静态存储位元的电路图。图中用插图的形式将DRAM位元与其进行对比。静态存储器的速度比动态存储器的速度要快。

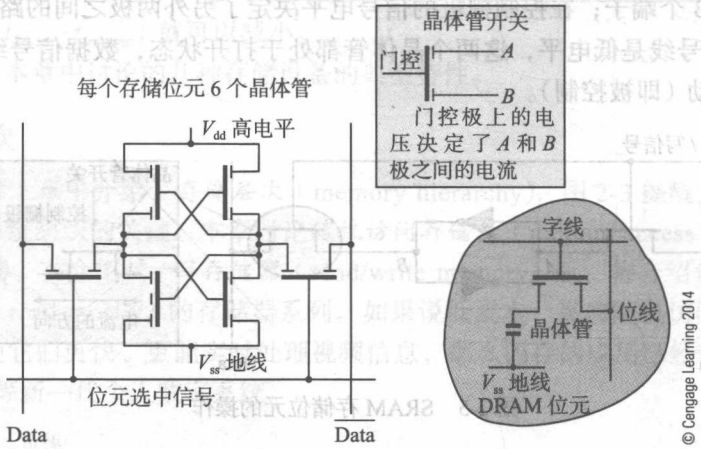


图2-6 由6个晶体管构成的SRAM存储位元与DRAM对比

半导体技术的进步总是在继续。2001年，日本富士通实验室的研究人员公布了一款由铟镓砷化物（与常见的半导体材料硅对比是一种很专业的物质）构成的单晶体管静态存储位元的设计，可以工作在77K（−196℃）。这样的温度对当前的实际应用来说都太低。它可以通过量子隧道（quantum tunneling）效应来存储数据。一些其他技术，如纳米管，可能是未来高密度、低功耗存储器的可能竞争者。

一种实用的半导体SRAM芯片由独立的存储位元阵列构成。图2-7给出了一个16位的存储阵列（真正的静态存储阵列可能包含2^{24}个位元）。4位存储器地址$A_0 \sim A_3$被分为行、列地址。二－四译码器对两位行地址进行译码后选中一行。

同时，列译码器选中4列之一。每行和每列的交叉处是存储位元，根据$\overline{\text{Write}}$信号的状态进入读或写周期。这个阵列说明存储器操作是以前面介绍的逻辑部件的形式进行的。存储阵列的实际实现将使用图2-6中所示的存储位元。

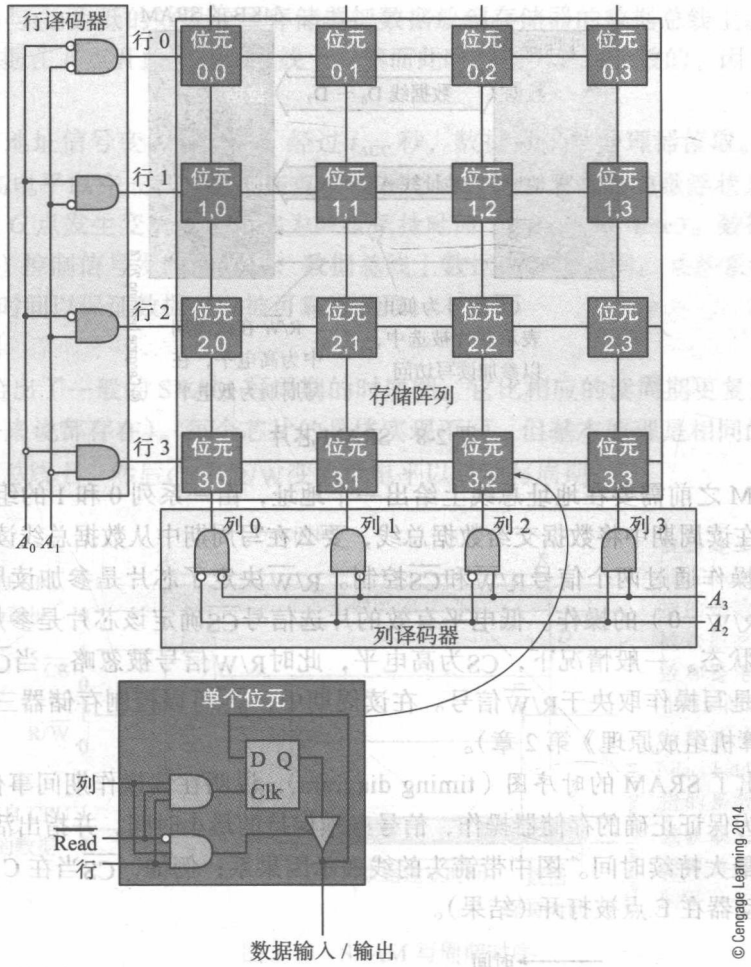


图 2-7 SRAM 阵列的组织

两全其美

Cyprus 半导体公司制作了一款 1Mb 的半导体 SRAM，组织成 128K 个字、每字 8 位或者 64 个字、每字 16 位。其访问时间为 25ns。该设备的主要特点是在每个静态位元中合并了非易失性部件，采用量子陷阱（quantum trap）技术来存储数据。当设备断电后，静态位元中的数据被存储在非易失性部件中，可以在加电后恢复。因此，该设备具有所有静态存储器的优点以及非易失性存储器的优点。

1. SRAM 存储系统

下面来看看如何使用静态存储器。RAM 芯片被组织成一位宽、半字节宽（4 位）、字节宽（8 位）或 16 位宽的设备。图 2-8 给出了一个容量为 512Kb 的 SRAM，它有 64K 字，每个字 8 位。它有 16 条地址线 A_0 ～ A_{15}，可以选择 2^{16}=64K 个位置，它还有 8 条数据线 D_0 ～ D_7 可以在一个读周期内将 8 位数据交给处理器，也可以在一个写周期内从处理器接收 8 位数据。

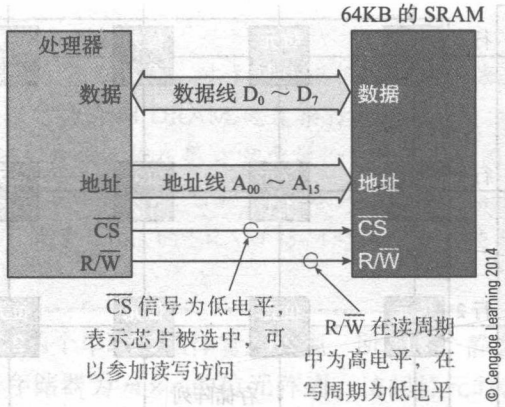
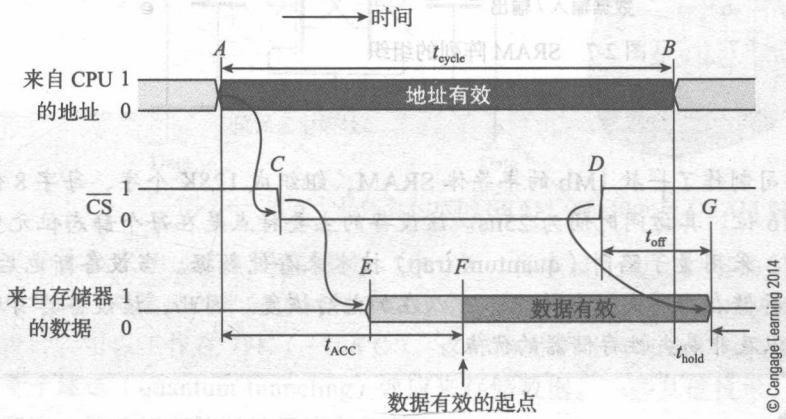


图 2-8 SRAM 芯片

访问 SRAM 之前需要在地址总线上给出一个地址，由一系列 0 和 1 的组合定义访问位置，然后要么在读周期中将数据交给数据总线，要么在写周期中从数据总线读取数据。

SRAM 的操作通过两个信号R/$\overline{W}$和$\overline{CS}$控制。R/$\overline{W}$决定了芯片是参加读周期（R/$\overline{W}$ =1）还是写周期（R/$\overline{W}$ =0）的操作。低电平有效的片选信号$\overline{CS}$确定该芯片是参加读或写访问，还是保持空闲状态。一般情况下，$\overline{CS}$为高电平，此时R/$\overline{W}$信号被忽略。当$\overline{CS}$为低电平时，芯片参与读还是写操作取决于R/$\overline{W}$信号。在读周期中，$\overline{CS}$可以控制存储器三态缓冲器的输出（参见《计算机组成原理》第 2 章）。

图 2-9 给出了 SRAM 的时序图（timing diagram），说明在读操作期间事件发生的顺序。时序图表明，为保证正确的存储器操作，信号必须保持的最小时间，并指出活动从开始到结束之间可能的最大持续时间。图中带箭头的线表示因果系；例如，$\overline{CS}$当在 C 点变低（原因）使数据总线驱动器在 E 点被打开（结果）。



这些参数给出了执行动作的时间或动作所需的时间。t_{cycle} 为完成读周期所需的最小时间。t_{ACC} 为访问时间，这是等待数据所必须花费的最长时间。t_{off} 为当$\overline{CS}$变高时，数据总线变为悬浮（float）状态所需的时间。t_{hold} 为地址改变后数据还能保持的时间

图 2-9 SRAM 读周期时序

假设最初（图的左边）$\overline{CS}$为高，R/$\overline{W}$在整个操作期间保持 1 状态。R/$\overline{W}$在图 2-9 中没有表示出来。

在图 2-9 中的 *A* 点，地址总线通过改变状态为当前读操作提供一个有效地址。这种状态的变化由地址线的交叉部分表示。一旦地址信号稳定下来，片选信号在 *C* 点指出开始进行存储器访问。由于R/$\overline{W}$为高电平，故为读操作周期。

R/$\overline{W}$为高与$\overline{CS}$为低的效果导致存储器把数据放到存储器的数据总线上。数据总线结束悬浮状态，数据在 E 点开始出现在总线上。然而此时数据仍然是无效的，因为数据的访问时间尚未开始。

在 F 点，地址信号变为有效后，经过 t_{ACC} 秒，数据可以被处理器读取。在 D 点，处理器将$\overline{CS}$变成高电平以完成读周期，在点 G 数据总线再次变成高阻的悬浮状态。从 B 点改变地址到数据在 G 点发生变化的时间称为*数据保持时间*（data hold time）。数据保持时间定义为地址和（或）控制信号发生改变后，数据总线上数据的保持时间。某些系统需要正的（非零）数据保持时间以保证数据可以被可靠地获取。

2. 写周期

图 2-10 给出了一般的 SRAM 写周期的时序图，它比相应的读周期更复杂（这情况对所有的存储设备来说都存在）。每个芯片的具体实现不同，但基本原理是相同的。要存储的数据先放到数据总线上，然后$\overline{CS}$和R/$\overline{W}$变为低电平以启动写周期。

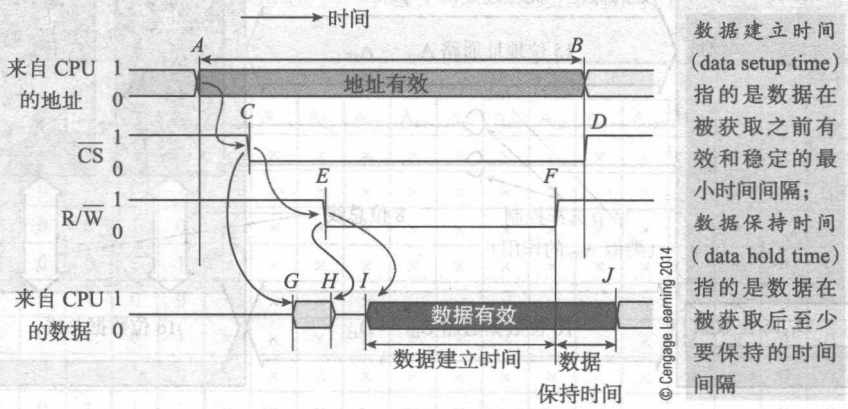


图 2-10　SRAM 写周期时序

图 2-10 中，地址总线在 A 点开始有效直到写周期在 B 点结束。假定$\overline{CS}$在 C 点变为低电平。此时，R/$\overline{W}$仍然为高电平，数据总线处于悬浮状态。

访问在 C 点开始，但对 RAM 而言，只要R/$\overline{W}$为高电平就表示是读周期。因此，在点 G，RAM 开始用读出的数据驱动数据总线。当然，此时的数据是无效的，因为数据的访问时间尚未开始。

在 E 点，处理器使R/$\overline{W}$变为低电平表明当前周期是写周期。存储器终止已经开始启动的读周期，并在点 H 停止用读出的数据驱动数据总线。可见，存储器在点 G 和 H 之间给出了假的数据。系统设计师必须保证，此时没有其他设备使用总线。

在点 I，处理器将要保存的数据放到数据总线上。写周期的关键点为 F 点，此时R/$\overline{W}$信号的上升沿表示开始捕获数据。数据必须在R/$\overline{W}$的上升沿之前有效，并保持到低电平变为高电平之后，此段时间为*数据保持时间*（data hold time）。真正 SRAM 的设计以$\overline{CS}$或R/$\overline{W}$中的任何一个变为高电平来终止写周期。

介绍完了 SRAM，自然就应该介绍 DRAM。但是在这之前，还应该顺便介绍影响所有存储器系统设计的两个问题，无论是静态 / 动态、读写 / 只读存储器。这两个问题就是字节 / 字控制以及地址译码的使用。

3. 字节 / 字控制

20 世纪 70 年代，微处理器都是按字节宽度设计的，数据总线为 8 位宽。地址总线为 16 位，使用 A_{15} ～ A_{00} 这些地址线可以选定 2^{16}=64K 个字节。当出现了 16 位处理器后，情况变得更复杂，此时需要访问字节和 16 位的字（记住，存储器是以字节编址的，无论字长是 1 个、2 个、4 个还是多个字节，都可以访问存储器中的一个字节）。

由于微处理器设计者既希望访问单个字节又希望具有访问 16 位的能力，故他们实现了字节控制机制允许访问单个字节或者选中的字。一种典型的机制是使用地址总线选中一个 16 位的字（或 32 位或 64 位），然后使用字节控制线（byte control line）选中该地址处的一个或多个字节。图 2-11 给出了一种可能的实现，通过地址线 A_{15} ～ A_{01} 来选中 2^{15} 个 16 位的字。

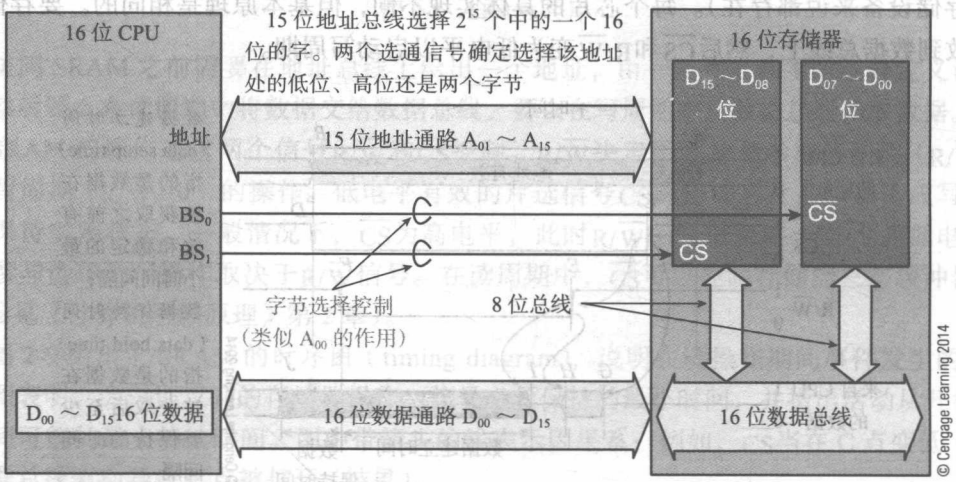


图 2-11　字节 / 字控制

地址线 A_{00} 没有使用，因为每个地址总是选中两个字节；即地址为 0,2,4,6,8,…,2^{15}。两个字节选通信号 BS_0 和 BS_1 控制选中字的奇数字节还是偶数字节，或者同时选中两个字节（参见表 2-2）。

表 2-2　存储器中的字节选择

BS_1	BS_0	操作	BS_1	BS_0	操作
1	1	无操作	0	1	选择高位字节 D_8 ～ D_{15}
1	0	选择低位字节 D_0 ～ D_7	0	0	选择两个字节 D_0 ～ D_{15}

存储器地址的范围

存储器设备的地址输入可以跨越由地址给出的每个地址位置；例如，如果存储器有 6 个地址引脚，其地址范围从 000000,000001,000010 一直到 111111。

为了避免繁琐的二进制运算，通常用十六进制来表示地址；例如，一个 6 位设备的地址空间可以表示为 00 ～ 2F。

假设计算机具有 64MB 的地址空间，由地址线 A_{00} ～ A_{25} 寻址；使用 8MB 的存储器模块，用 A_{00} ～ A_{22} 寻址。即存储模块内部使用 A_{00} ～ A_{22} 寻址，A_{23} ～ A_{25} 用来选择 8 个 8MB 的存储模块中的一个（8×8MB=64MB）。8 个模块的地址空间分别为：

000 0000 to 03F FFFF
040 0000 to 07F FFFF
080 0000 to 0BF FFFF
0C0 0000 to 0FF FFFF
100 0000 to 13F FFFF
140 0000 to 17F FFFF
180 0000 to 1BF FFFF
1C0 0000 to 1FF FFFF

如果希望定位地址空间从 0C0 0000 至 0FF FFFF 这 8MB 的块，地址总线的最高 3 位将是 011。即 A_{25}、A_{24} 和 A_{23} 必须是 0、1、1 才可以选择块中的某个地址。使用简单的逻辑元件就可以完成该地址的译码操作。

假设某基于 8 位微处理器的控制器具有 16 位地址总线、两个 8K×8 位的闪存加上两个 4K×8 位的 SRAM。需要使用高位地址访问闪存（保存中断向量表）和低位地址访问 SRAM。下面就是该方法对应的译码表。R1、R2、S1 和 S2 是用于选择 4 个存储部件的信号。可见，R1 与地址 0000-0FFF 对应，R2 与地址 1000-1FFF 对应，S1 和 S2 对应地址 C000-DFFF 和 E000-FFFF。

	地址线																设备选择			
	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}	R1	R2	S1	S2
0000-0FFF	0	0	0	0	×	×	×	×	×	×	×	×	×	×	×	×	1	0	0	0
1000-1FFF	0	0	0	1	×	×	×	×	×	×	×	×	×	×	×	×	0	1	0	0
2000-2FFF	0	0	1	0	×	×	×	×	×	×	×	×	×	×	×	×	0	0	0	0
3000-3FFF	0	0	1	1	×	×	×	×	×	×	×	×	×	×	×	×	0	0	0	0
4000-4FFF	0	1	0	0	×	×	×	×	×	×	×	×	×	×	×	×	0	0	0	0
5000-5FFF	0	1	0	1	×	×	×	×	×	×	×	×	×	×	×	×	0	0	0	0
6000-6FFF	0	1	1	0	×	×	×	×	×	×	×	×	×	×	×	×	0	0	0	0
7000-7FFF	0	1	1	1	×	×	×	×	×	×	×	×	×	×	×	×	0	0	0	0
8000-9FFF	1	0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	0	0	0	0
A000-BFFF	1	0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	0	0	0	0
C000-DFFF	1	1	×	×	×	×	×	×	×	×	×	×	×	×	×	×	0	0	1	0
E000-FFFF	1	1	×	×	×	×	×	×	×	×	×	×	×	×	×	×	0	0	0	1

4. 地址译码

本节将简介地址译码技术，它是将地址映射到处理器物理地址空间中的方式。这里使用《计算机组成原理》第 2 章中的概念（二进制、十六进制以及简单的逻辑）。因为地址译码多是嵌入式系统设计师而不是计算机设计师要考虑的事情，有些读者可能希望忽略这部分内容。

在所有可能中最好的情况是，计算机具有 n 位地址总线可以访问 2^n 个字，同时存储部件有 n 位地址输入和 2^n 个存储器位置。实践中，实际系统可能有几种不同的存储设备连接到处理器的总线。如果计算机处理器有 32 位地址总线，可访问 2^{32} 个字节，其存储模块（存储器的基本元件为一个芯片，而存储模块是一个包含多个存储芯片的电路板）提供 2^{29} 个字节（512MB）的存储；即需要 $2^{32}/2^{29}=2^3=8$ 个模块来满足处理器的空间需求。此时需要一种将存储模块的地址空间映射到处理器地址空间的方法。称为地址译码器的电路用来检测对特定存储模块的访问，一旦 CPU 产生的地址落在某存储模块的地址空间中，则将该模块对应的$\overline{CS}$信号变为有效。

图 2-12 说明了为什么需要地址译码。假设处理器使用地址线 A_{00} ～ A_{31} 访问 4096MB 的地址空间；即 2^{32}=4096MB。该系统采用 3 个 512MB 的存储模块，每个模块由地址线 A_{00} ～ A_{28} 访问。

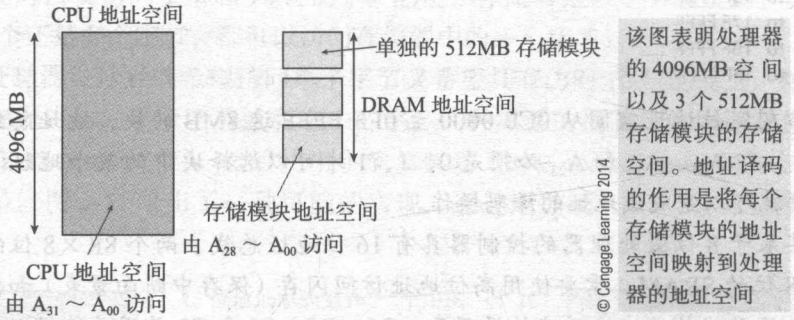


图 2-12 处理器地址空间和存储器空间之间的关系

图 2-13 显示了这 3 个 512MB 的模块是如何映射到处理器的存储空间中的。虽然已将模块映射到存储空间相邻的区域，但没有规定物理存储器必须被映射到相邻的块。当然，如果存储区域不连续，程序员或操作系统在加载程序和数据时就需要特别小心。模块 1 分配的地址空间为 0000 0000 到 1FFF FFFF，模块 2 的为 2000 0000 到 3FFF FFFF，模块 3 的为 4000 0000 到 5FFF FFFF。当然，每个 512MB 的块都是由 A_{00} ～ A_{28} 访问。

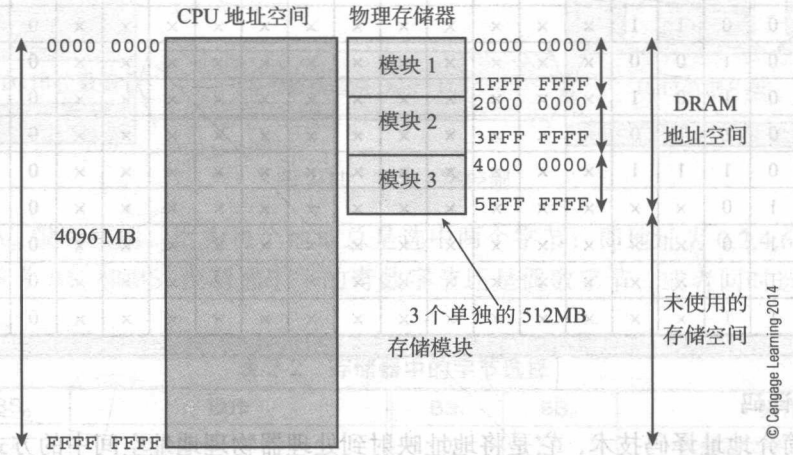


图 2-13 将存储器空间映射到处理器空间

图 2-14 显示了如何设计执行存储模块映射到处理器地址空间的逻辑。3 个模块中的每个模块的 29 个地址线都与处理器 A_{00} ～ A_{28} 连接在一起。用从处理器那里剩下的 3 根地址线 A_{29} ～ A_{31} 选择某存储模块，即这些地址线产生的信号可以选中对应的存储模块。

第一块存储器 512MB 空间的地址为 $0000\ 0000_{16}$ 至 $1FFF\ FFFF_{16}$。该模块在 $A_{31}A_{30}A_{29}$=000 时被选中。同样，第二块存储器空间的地址为 $2000\ 0000_{16}$ 至 $3FFF\ FFFF_{16}$，该模块在 $A_{31}A_{30}A_{29}$=001 时被选中。

这里不再进一步探讨地址译码，因为这是系统设计者和电子工程师要考虑的问题。地址译码可以用来匹配存储部件和 CPU 的地址空间，它还可以做得更多。例如，可以在多个存储器模块之间实现切换来保证任务在物理上是彼此分离的（一种内存管理的形式）。

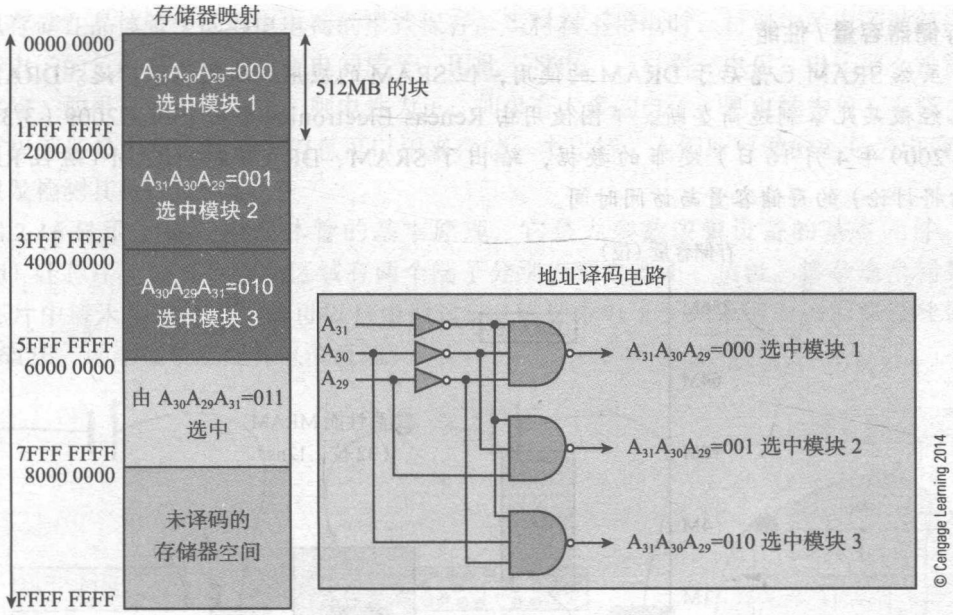


图 2-14　地址译码器的实现

2.2.2　交叉存储器

如果存储器的访问时间为 t_{acc}，则不能减少其访问时间。但是可以通过交叉（interleaving）来减少有效存取时间（effective access time）。若某存储位置在 T_0 时刻开始访问，则数据在 t_{acc} 时间后变为有效。在同一存储模块内另一个位置处的数据至少要再等 t_{acc} 时间后才能获取。但是，可以同时访问其他的存储模块；即并行访问多个存储器模块。

图 2-15 说明了这种交叉的效果，图中两个存储体并行排列。如果在 T_0 时刻访问第一个存储体，数据在 t_{acc} 时间后变为有效。如果在 T_0+t_{cyc}（其中 t_{cyc} 为时钟周期时间）时刻访问第二个存储体，则第二个数据元素在 $T_0+t_{cyc}+t_{acc}$ 时间后有效。如果 $t_{cyc} < t_{acc}$，则可以比在没有交叉时更早地完成第二次访存任务。只有在一个存储体访问数据时能够产生另一个存储体中操作数的地址时，有效的交叉才能实现。

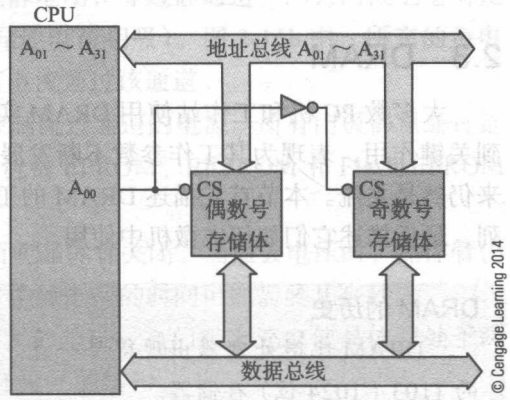
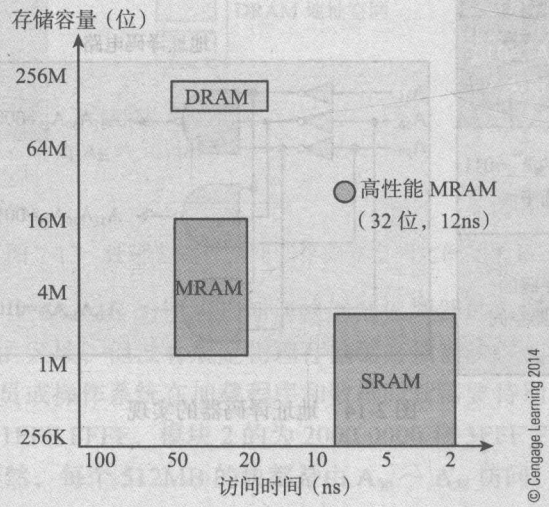


图 2-15　交叉存储器

存储交叉技术被称为低位交叉，其中存储地址的低位用来选择存储体。假设处理器有 32 位地址总线和 64 位数据总线。地址位 A_{03} ~ A_{31} 用来选择存储器中的一个字，地址位 A_{00} ~ A_{02}（通过字节选通信号）用来选中当前字中的一个或多个字节。使用地址位 A_{04} 执行存储体选择，所有奇数号地址的数据字从一个存储体中取出，而所有偶数号地址的数据字从另一个存储体中取出。如果使用两个地址位来实现交叉，则可以实现 4 个存储体并行访问。在现代微机中，双通道或三通道内存的原理同交叉的原理相同，允许同时访问 2 ~ 3 个 DRAM 存储模块。

存储器容量 / 性能

虽然 SRAM 已落后于 DRAM 的使用，但 SRAM 的应用领域仍然很广泛。DRAM 生产已经被某几家制造商垄断。下图使用由 Reneas Electronics 公司 ISSCC2009（3/3）85 卷（2009 年 4 月 16 日）发布的数据，给出了 SRAM、DRAM 和 MRAM（磁性 RAM，稍后将讨论）的存储容量与访问时间。



下面将介绍 DRAM 的工作原理。通过几代 DRAM 设备的不断发展，它们基于相同的基本工作原理，但使用不同的技术来降低平均访问时间。下面将介绍基本的 DRAM，讨论其不同的变种，如页模式 DRAM、半字节模式 DRAM、EDO 存储器、SDRAM 和 DDR DRAM。

2.3 DRAM

大多数 PC 机和工作站使用 DRAM 实现主存储器，它已经在高性能计算机的发展中起到关键作用，表现为其工作参数不断发展变化且出现新的变种。我认为 DRAM 在不久的将来仍然是主流。本节首先描述 DRAM 的工作原理，接着介绍它的时序，然后介绍 DRAM 系列，最后描述它们如何在微机中使用。

DRAM 的历史

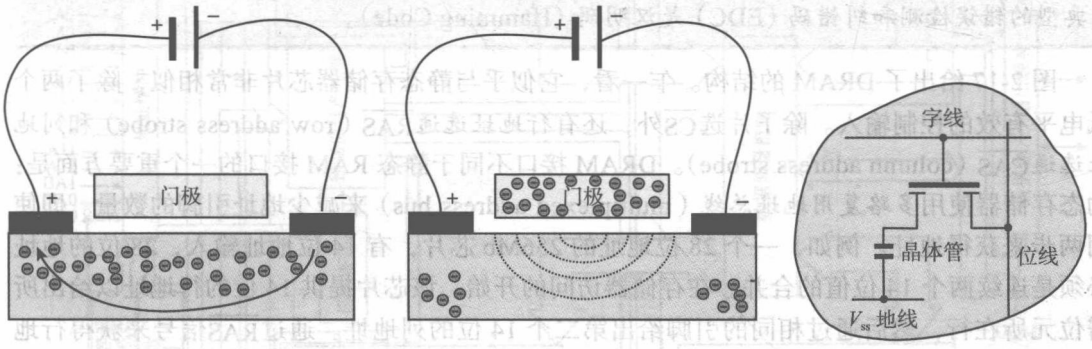
DRAM 比微处理器出现得早一点。事实上，世界上第一款商用 DRAM 芯片是 Intel 的 1103（1024 位）存储器。

使用单一晶体管实现一位动态存储器的想法可以追溯到 1966 年 R.H.Denning 在 IBM 的工作。第一款 DRAM 使用 P 沟道金属氧化物半导体（PMOS）技术构造，它现在已经过时了。几年后，4K 位 DRAM 采用 N 沟道金属氧化物半导体（NMOS）技术构造。互补金属氧化物半导体（CMOS）技术的引进是一大突破，因为它大大降低了功耗。目前的 DRAM 仍使用 CMOS 技术。

动态随机访问存储器（dynamic random access memory，DRAM）的工作原理很简单。

数据以存储在晶体管中的静电电荷的形式保存。当材料不带电时，材料中的电子数等于质子数；即电子的负电荷抵消了带正电的质子，因此不带电。当材料充电后，电子要么被添加要么被移除。如果有太少的电子，则电性为正；如果有太多的电子，则电性为负。在微小的电容上有极性还是没有极性这个信息可以用来存储一个比特。人们所要做的就是为电容充电、放电以及检测其是否带电。

图 2-16 显示了场效应晶体管的基本原理，它是大多数逻辑设备的基本元件。掺杂（doped）硅芯片上的一个微小区域有两个端子分别连到电池正、负极。掺杂这个词意味着在硅芯片中掺入杂质使电子流可以自由通过硅（纯硅中电子受到束缚而不能穿过硅材料）。图 2-16a 中，两个端子之间有电流流动。



a）门极上没有充电　　b）门极上有负电　　c）DRAM 位元

图 2-16　带电门极对电子流的影响

图 2-16 中，在硅区域上方的导电体被标记为门极。假设其与图 2-16a 中两个端子之间硅通道中的电子流是绝缘的。

在图 2-16b 中，对门极充上负电荷。电荷产生静电场，穿过硅通道。因为门极上电荷是负的，故通道中的电子被排斥（同性电荷相斥，异性电荷相吸）。图 2-16b 中，所充的负电荷十分强大，通道处于夹断（pinched）状态，没有电流通过该通道。

通过对门极充负电荷还是不充负电荷，可以控制流过通道的电流。所有门极都是通过这种方式来切断电子流的。它还可用于 DRAM 以及包括 EPROM、EEPROM 和 Flash EPROM 在内的大部分时间为读的存储器。

如果将负电压加到门极上以便对其充电，从而使晶体管关闭。当撤去电压时，晶体管仍然不能导通直到电荷消除。如果门极与通道绝缘，消除电荷的时间可能需要几毫秒。

在电容中保存电荷是形成单晶体管 DRAM 位元的基础。在门极上充电使晶体管处于两种状态之一。图 2-16c 展示了一个单晶体管 DRAM 位元的结构。

保存的电荷最终会泄漏，存储在位元中的任何数据都会丢失，使所有位元处于相同的状态。为了使保存的电荷具有记忆效应，实际的存储器必须每隔几毫秒就读取晶体管的状态然后将其写回晶体管。这种操作称为刷新（refreshing）。

第一代 DRAM 存储器的刷新是一场噩梦，需要复杂的电路和定时器，因此开销很大。只要可能，设计师就会使用 SRAM 来避免使用 DRAM 进行系统设计带来问题。今天情况已经好转，因为将刷新逻辑设计在 DRAM 芯片中使 DRAM 的刷新要求大大简化。

辐射的威胁

DRAM 会受所谓的 α 粒子问题（alpha-particle problem）的影响。位元存储的电荷非常少，α 粒子（由原子放射性衰变产生）通过存储单元时能够引起足够的电离，从而导致所存储的数据损坏。α 粒子产生的是软错误（soft error），这是因为位元失去了存储的数据，但并没有被永久损坏。α 粒子的污染主要发生在封装材料中。

半导体制造商通过仔细控制封装芯片材料的质量来最小化该问题。然而，由于宇宙辐射导致的软错误是不可能被禁止的。软错误率不能被降低到零，在 1Gbit 的 DRAM 存储器中每小时软错误的量级是 2^{-11} 个（即大约与赢得彩票的概率相同）。

系统需要纠错存储器来提高可靠性，其中每个字存储一个附加位用于修复软错误。典型的错误检测和纠错码（EDC）是汉明码（Hamming Code）。

图 2-17 给出了 DRAM 的结构。乍一看，它似乎与静态存储器芯片非常相似，除了两个低电平有效的控制输入。除了片选$\overline{CS}$外，还有行地址选通$\overline{RAS}$（row address strobe）和列地址选通$\overline{CAS}$（column address strobe）。DRAM 接口不同于静态 RAM 接口的一个重要方面是：动态存储器使用多路复用地址总线（multiplexed address bus）来减少地址引脚的数量。即使用两步来获得地址。例如，一个 28 位地址的 256Mb 芯片，有 14 位地址输入，28 位的地址必须是连续两个 14 位值的合并。在存储器访问的开始，该芯片提供 14 位的行地址以给出所需位元所在行，然后通过相同的引脚给出第二个 14 位的列地址。通过$\overline{RAS}$信号来获得行地址，使用与$\overline{CAS}$信号来获得列地址。然后才能访问给定行列位置的位元。

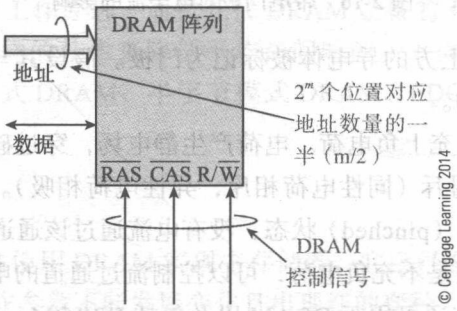


图 2-17 DRAM 存储部件的输入与输出

将地址总线从 28 位减少到 14 位需要两个阶段来锁存地址。由行地址选通$\overline{RAS}$锁存 14 位行地址，列地址选通$\overline{CAS}$锁存 14 位列地址。地址复用以及$\overline{RAS}$和$\overline{CAS}$的选通控制由片外逻辑提供。

图 2-18 给出了 DRAM 存储器的内部结构。最需要注意的一点是其中包含不止一个存储阵列。商用芯片通常由 4 个存储阵列或存储体组成。

图 2-19 给出了某 4MB 的 DRAM 的结构，其地址为 $A_{00} \sim A_{21}$，由 32 个 1Mb 的 DRAM 芯片构成。来自计算机的地址线 $A_{02} \sim A_{21}$ 可以选择 2^{20} 个 32 位的字。DRAM 存储子系统使用多路复用器从地址总线上选择行或列地址。控制单元根据 CPU 的控制信号产生 DRAM 的$\overline{RAS}$、$\overline{CAS}$和$\overline{W}$信号。

介绍了 DRAM 的基本概念后，下一步将介绍它的时序，因为时序是理解所有 DRAM 后期变化的关键。

图 2-18 DRAM 芯片结构

图 2-19 DRAM 系统的结构，不包含字节选择逻辑

2.3.1 DRAM 时序

本节首先介绍20世纪80年代经典的DRAM时序，所有现代DRAM时序均可由其变化得到。虽然DRAM存储位元是由一个晶体管构成这一属性基本没有变化，了解DRAM时序的基本属性是为了理解，DRAM是如何一代一代变得更快的。图2-20给出一个简化的DRAM芯片的读周期时序图。假定1Mb的芯片组织具有2^{20}个地址，每个地址1位。

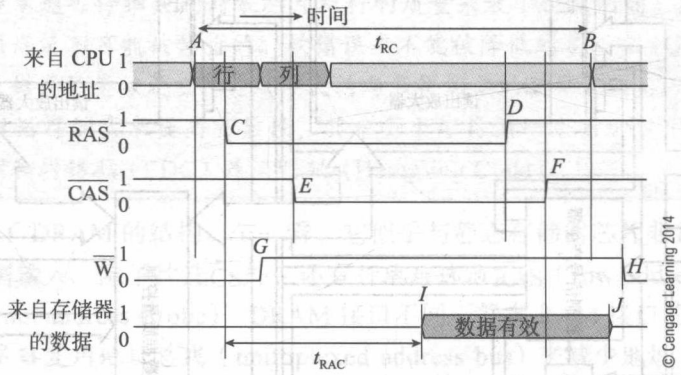


图2-20 DRAM读周期时序

来自CPU的20位地址A_{00}～A_{19}被送到一个多路复用器，要么选择A_{00}～A_{09}（行地址）要么选择A_{10}～A_{19}（列地址）。CPU地址多路复用器（MPLX）的10个输出连接到DRAM地址输入A_0～A_9。

图2-20中的读周期从*A*持续到*B*，最短持续时间为t_{RC}，即读周期时间（read cycle time）。静态存储器的存储周期与访问时间相同；动态存储器的存储周期比其访问时间要长。因为在当前访问结束后，在两次访问之间，DRAM需要执行被称为预充电（pre-charging）的内部操作，因此不能开始新的访问。

DRAM 参数

本章使用如下的一些时序参数：

t_{RC}	读周期最大时间
t_{RAC}	$\overline{RAS}$变为低电平至数据有效的时间
t_{ASR}	行地址建立最小时间
t_{RAH}	行地址保持最小时间
t_{ASC}	列地址建立最小时间
t_{CAH}	列地址保持最小时间
t_{RCD}	$\overline{RAS}$变为低电平至$\overline{CAS}$变为低电平的最小时间
t_{CAC}	$\overline{CAS}$变为低电平至数据有效的最小时间
t_{OFF}	$\overline{CAS}$变为高电平至数据失效的最小时间

读周期的第一步是向芯片输入来自CPU的低10位地址A_0～A_9。然后，在点*C*，行地址选通（row address strobe，$\overline{RAS}$）变为低电平有效，将行地址保存到芯片内部的锁存器。一旦获得行地址，从处理器来的低位地址就是多余的，周期剩余部分不再需要它。相比之下，SRAM的地址必须在整个读或写周期中保持稳定。

来自 CPU 的高 10 位地址接着被交给存储器的地址输入，然后列地址选通（column address strobe，$\overline{CAS}$）在 E 点变为低电平有效，锁存列地址。此时，所有 20 位地址都已交给存储器，系统可以改变地址总线的内容，因为 DRAM 已获得了地址。

一旦$\overline{CAS}$变为低电平，根据地址选中的存储单元将数据放到其数据输出端口，允许 CPU 读取。读周期结束时，$\overline{CAS}$变为高电平无效，数据总线被关闭，变为悬浮状态。$\overline{RAS}$和$\overline{CAS}$可以同时、也可以次序任意变为高电平。无论谁先变为高电平都无所谓，只要其他的时序满足要求。要使对 DRAM 的解释更容易理解，需要将它分为几个部分，并从地址引脚的作用开始讨论。

详细的 DRAM 地址时序如图 2-21 所示。行地址必须在$\overline{RAS}$信号下降沿到来之前稳定至少 t_{ASR} 时间（即行地址建立时间）。由于 t_{ASR} 的最小值可能是 0ns，此时行地址的建立时间为零，不需要在$\overline{RAS}$信号下降沿到来之前有效。最坏情况下，它必须在$\overline{RAS}$信号下降沿到来之前有效。当$\overline{RAS}$已经变为低电平，行地址必须稳定 t_{RAH} 时间不改变，即行地址保持时间。保持时间限制了在此之后列地址才可以被多路复用到芯片上的地址引脚上。

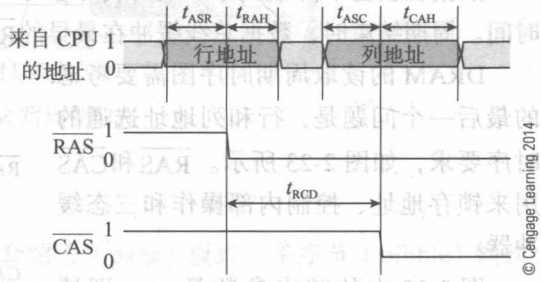


图 2-21　DRAM 的地址时序

一旦行地址保持时间已经得到满足，列地址可以复用到存储器的地址引脚，$\overline{CAS}$就可以变为低电平。列地址的建立时间 t_{ASC} 通常最小为 0ns；即$\overline{CAS}$可以在列地址有效的同时变为低电平。$\overline{CAS}$变为低电平有效后，列地址必须稳定 t_{CAH} 时间才可以改变，这就是列地址保持时间。一旦 t_{CAH} 已经得到满足，地址总线在当前访问中就没有作用了。

行地址必须在行地址选通信号下降沿到来之前保持有效 t_{ASR} 时间，且在下降沿之后保持有效 t_{RAH} 时间。同样，列地址必须在列地址选通信号下降沿到来之前保持有效 t_{ASC} 时间，且在下降沿之后保持有效 t_{CAH} 时间。$\overline{RAS}$和$\overline{CAS}$下降沿之间的最小时间为 t_{RCD}，该时间由行地址保持时间、多路复用器切换时间和列地址建立时间组成。

图 2-21 中的一个重要参数是 t_{RCD}，即由其他参数决定的行列选通时间。t_{RCD} 的最小值由行地址保持时间、多路复用器切换时间和列地址建立时间决定。t_{RCD} 的最大值为伪最大值（pseudomaximum），此时超出了存储器的访问时间。t_{RCD}（max）、t_{RAC} 和 t_{CAC} 之间的关系为：t_{RCD}（max）$=t_{RAC}-t_{CAC}$。

通过$\overline{RAS}$和$\overline{CAS}$就可以锁存地址，芯片数据引脚上就会出现数据，如图 2-22 所示。为清晰起见，图 2-22 中只画出了$\overline{RAS}$、$\overline{CAS}$以及数据信号。假设$\overline{W}$信号在读周期内为高电平，地址建立时间、保持时间和所有相关的参数都得到了满足。

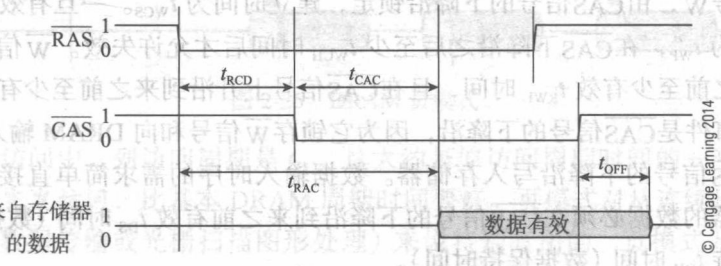


图 2-22　DRAM 读周期中的数据时序

$\overline{RAS}$下降沿之后，数据引脚上的数据在 t_{RAC} 时间后开始有效，这就是行地址选通后的访问时间。可见，行访问时间只有在其他条件都满足了之后才能实现。列地址选通具有两个功能：锁存存储阵列相应的列地址，以及打开数据输出缓冲。$\overline{CAS}$下降沿之后，数据至少在 t_{CAC} 时间内不可用，这是列地址选通后的访问时间。

当$\overline{CAS}$在读周期结束后变为高电平时，数据总线被关闭，并悬浮 t_{OFF} 时间（输出缓冲关闭延迟）。$\overline{RAS}$在读（或写）周期结束时没有作用。可以在$\overline{CAS}$之前或之后改变$\overline{RAS}$，只要它的时序要求得到满足。

数据在$\overline{CAS}$下降沿到来以后保持不多于 t_{CAC} 时间，且在$\overline{RAS}$下降沿之后保持不多于 t_{RAC} 时间。周期结束时，数据总线缓冲在最早的$\overline{RAS}$和$\overline{CAS}$上升沿 t_{OFF} 时间内被关闭。

DRAM 的读取周期时序图需要考虑的最后一个问题是，行和列地址选通的时序要求，如图 2-23 所示。$\overline{RAS}$和$\overline{CAS}$用来锁存地址、控制内部操作和三态缓冲器。

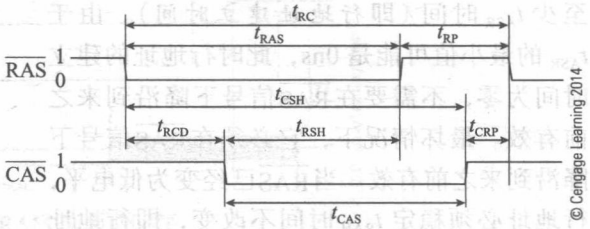


图 2-23 DRAM 行和列地址选通的详细时序

图 2-23 中的基本参数是 t_{RC}，即读周期时间，这是连续的存储器周期之间最短的持续时间。在每个读周期中，$\overline{RAS}$选通必须至少保持 t_{RAS} 时间（行地址选通脉冲宽度）。典型的最大值为 10 000ns，这是由于设备需要刷新且不能出现问题导致的，该时间是微处理器读周期的多倍。在$\overline{RAS}$失效后，它必须保持高电平至少 t_{RP} 时间，这是行地址选通预充电时间（动态存储器芯片的相关内部操作）。时序的最后一个约束为$\overline{RAS}$相对$\overline{CAS}$的保持时间 t_{RSH}。$\overline{RAS}$必须在$\overline{CAS}$有效后保持低电平 t_{RSH} 时间。

列地址选通时序的要求与行地址选通的类似。$\overline{CAS}$必须有效至少 t_{CAS} 时间，在下一个$\overline{RAS}$下降沿到来之前必须失效至少 t_{CRP} 时间，且从当前$\overline{RAS}$下降沿开始至少保持 t_{CSH} 时间有效。

写周期时序

DRAM 存储器的写周期比相应的读周期更复杂，因为对其$\overline{W}$和数据输入信号提出了严格的要求。通过对读周期时序图的介绍，这里不需要重复介绍相同的过程。图 2-24 给出了简化的 DRAM 存储器写周期的时序图。这是一个提前写周期（early write cycle），因为$\overline{W}$信号在$\overline{CAS}$变为低电平之前有效。某些 DRAM 的写周期中，$\overline{W}$信号是在$\overline{CAS}$变为低电平之后才有效的。读与写周期中的$\overline{RAS}$、$\overline{CAS}$以及地址输入的时序要求是相同的。

写命令信号$\overline{W}$，由$\overline{CAS}$信号的下降沿锁定，建立时间为 t_{WCS}。一旦有效，$\overline{W}$为低电平的最小持续时间为 t_{WP}，在$\overline{CAS}$下降沿之后至少 t_{WCH} 时间后才允许失效。$\overline{W}$信号必须在$\overline{RAS}$信号上升沿到来之前至少有效 t_{RWL} 时间，且在$\overline{CAS}$信号上升沿到来之前至少有效 t_{CWL} 时间。写周期中的关键事件是$\overline{CAS}$信号的下降沿，因为它锁存$\overline{W}$信号和向 DRAM 输入的数据。

数据在$\overline{CAS}$信号的下降沿写入存储器。数据输入时序的需求简单直接，只涉及 3 个参数。写入存储器的数据必须在$\overline{CAS}$信号的下降沿到来之前有效 t_{DS} 时间（数据建立时间），且在其后继续维持 t_{DH} 时间（数据保持时间）。

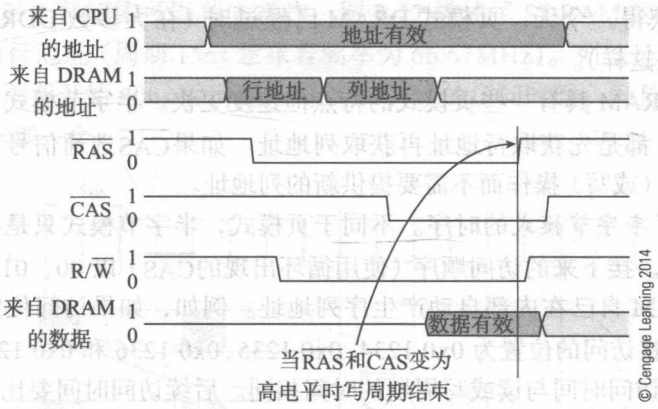


图 2-24 DRAM 写周期时序

2.3.2 DRAM 技术的发展

前面已经介绍了基本的 DRAM。下面开始介绍页（page）模式、半字节（nibble）模式、静态列（static column）模式等改进的 DRAM。这些变化仍然利用行列地址输入多路复用，同时克服了访问之间预充电造成一些限制。

页模式允许快速访问给定行中的任意列位置，如图 2-25 所示。假设某 1Mb 的 DRAM 处于一个正常的读或写周期中。首先给定 10 位行地址，$\overline{RAS}$信号变为低电平有效将其锁存，从而选中一行。其次为列地址，然后是$\overline{CAS}$信号有效，接着访问给定位置。页模式通过给出连续的$\overline{CAS}$脉冲，且在每个$\overline{CAS}$选通信号下降沿锁存新的列地址来连续访问同一行中的数据。

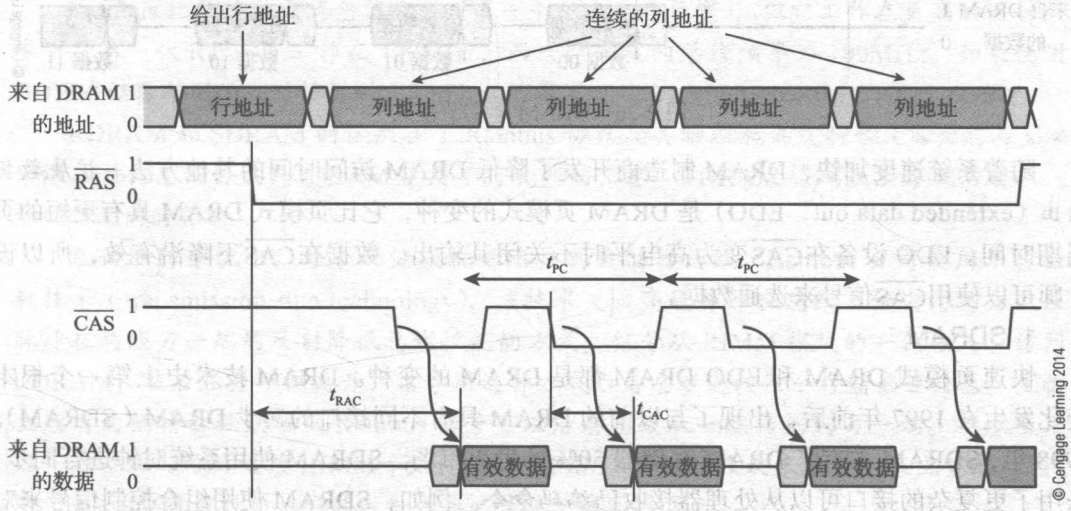


图 2-25 DRAM 页模式

连续页模式访问中，列访问周期是 t_{PC}，这大约是原访问周期时间的一半。页模式允许对单一行地址的突发访问，比基本 DRAM 周期时间要好。页模式对从连续地址访问一批数据（例如，成块数据传输或光栅扫描图形处理）来说特别有用的。页模式及其他突发模式可用符号 4:2:2:2 来描述，这意味着第一个访问延迟为 4 个周期，以后的数据元素只需要经

过两个周期即可获得。今天，页模式DRAM已经过时（在大多数DDR2和DDR3之前的DRAM变种中也是这样）。

半字节模式DRAM具有一些页模式的特点但速度更快。半字节模式访问开始时与普通DRAM访问相同，都是先获取行地址再获取列地址。如果$\overline{CAS}$选通信号重复出现，则对连续4个位置进行读（或写）操作而不需要提供新的列地址。

图2-26给出了半字节模式的时序。不同于页模式，半字节模式只是在突发访问开始时锁存单一的列地址。接下来的访问顺序（使用循环出现的$\overline{CAS}$）以00、01、10、11、00、10的顺序出现。DRAM自己在内部自动产生序列地址。例如，如果访问位置0x0 1234，然后循环出现的$\overline{CAS}$，将访问的位置为0x0 1234，0x0 1235，0x0 1236和0x0 1237（按顺序访问）。第一个半字节模式访问时间与读或写周期的时间相同。后续访问时间要比普通的访问周期的一半还要少。

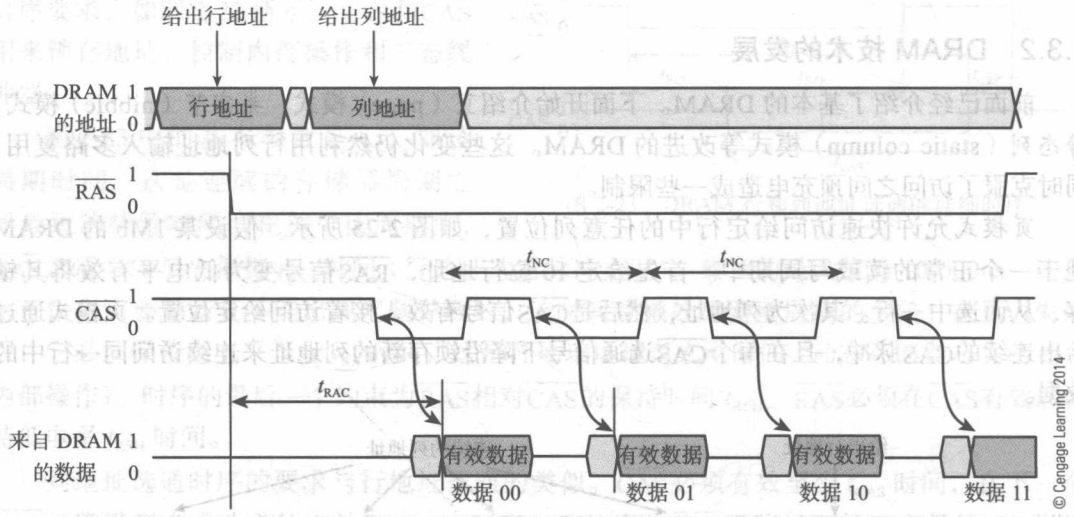


图2-26 DRAM半字节模式

随着系统速度加快，DRAM制造商开发了降低DRAM访问时间的其他方法。扩展数据输出（extended data out，EDO）是DRAM页模式的变种，它比页模式DRAM具有更短的页周期时间。EDO设备在$\overline{CAS}$变为高电平时不关闭其输出。数据在$\overline{CAS}$下降沿有效，所以设计师可以使用$\overline{CAS}$信号来选通数据。

1. SDRAM

快速页模式DRAM和EDO DRAM都是DRAM的变种。DRAM技术史上第一个根本变化发生在1997年前后，出现了与以前的DRAM具有不同接口的同步DRAM（SDRAM）。1998年，SDRAM占有了DRAM存储器50%的世界市场。SDRAM使用系统时钟进行同步，采用了更复杂的接口可以从处理器接收已编码命令。例如，SDRAM使用组合控制信号来对命令进行编码，如读、写或预充电。SDRAM访问时间与DRAM系列其他成员的访问时间类似；然而，它的突发访问时间却相当短。

控制信号和指令在时钟的上升沿锁存，这样做简化了系统的设计。SDRAM中的控制寄存器定义了它的操作参数，如突发长度（即每个读或写周期访问的字数）。也就是，SDRAM是一种操作参数可以面向特定系统进行调整的可编程装置。

SDRAM 具有多个存储体可以独立操作。图 2-27 描述了 SDRAM 的读周期。在时钟的上升沿获取命令和行地址（周期 15ns 意味着频率为 66.67MHz）。列地址在下一个时钟上升沿读取。数据在读周期开始 60ns 后出现，类似 DRAM 的访问时间。但是，接下来的 3 个元素每隔 15ns 就会出现一个。

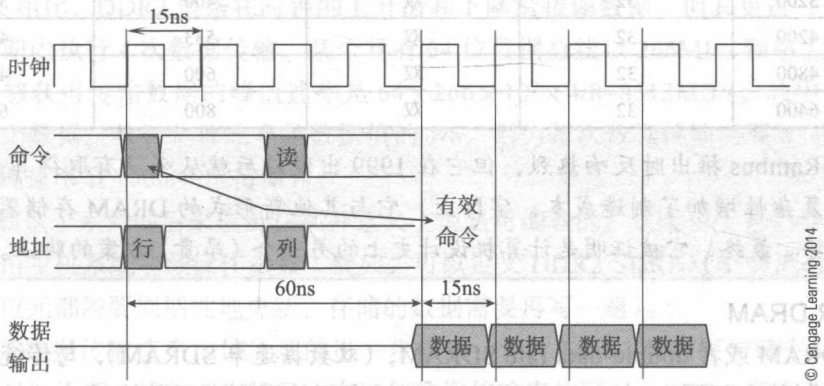


图 2-27　基本的 SDRAM 读周期

Rambus RAM

将 Rambus® 放在这里讨论是因为它代表了存储器技术发展的一种途径，但也被证明是一条死胡同。RDRAM 规范由 Rambus 公司与 Intel 通力合作、为提升高性能系统中 RDRAM 技术的性能而创建。RDRAM 是一种系统的方法，它涉及存储器技术和总线技术，因此叫作 Rambus。缩写 RDRAM 代表 Rambus DRAM，而存储模块被称为 RIMM（Rambus Inline Memory Module）

Rambus 比传统的数据总线要窄（第一个版本是 8 位宽），但它工作在更高的时钟频率下。第一代 Rambus 工作在 400MHz，而当时微机的总线频率为 100MHz。如果使用 DDR 存储器相同的双时钟机制，Rambus 的带宽可达 2×400MHz×16 位 /8=1.6GB/s。

RDRAM 和 SDRAM 的区别在于 Rambus 协议及其物理和电气特性（本文将在第 4 章深入讨论总线协议）。DRAM 复用了行和列的信息。RDRAM 将地址线分成两组，行和列信息同时传输。

在传统的系统中，每个存储模块都连接到存储器总线。Rambus 公司使用了传输线技术（transmission line technology），该技术关心高速脉冲在总线上传输的行为以及脉冲在总线另一端的反射降低总线性能的方式。信号从 RIMM 模块的一侧输入，访问 RDRAM，然后在另一侧离开，信号用这种方式沿着总线通过每个存储器连接器。主板上所有的插槽都必须确保 Rambus 信号可以沿着总线从一端传播到另一端。可以使用以一种特殊的称为连续 RIMM（continuity RIMM）的通过模块来填补主板上空的插槽。Rambus 的电气设计允许它的时钟频率比传统总线高很多。

下表给出了 Rambus 的位宽、通道数以及与数据传输带宽相对应的时钟频率。

类型	位宽	通道数	时钟频率 /MHz	带宽 /MB/s
PC600	16	单	300	1200
PC700	16	单	355	1420
PC800	16	单	400	1600

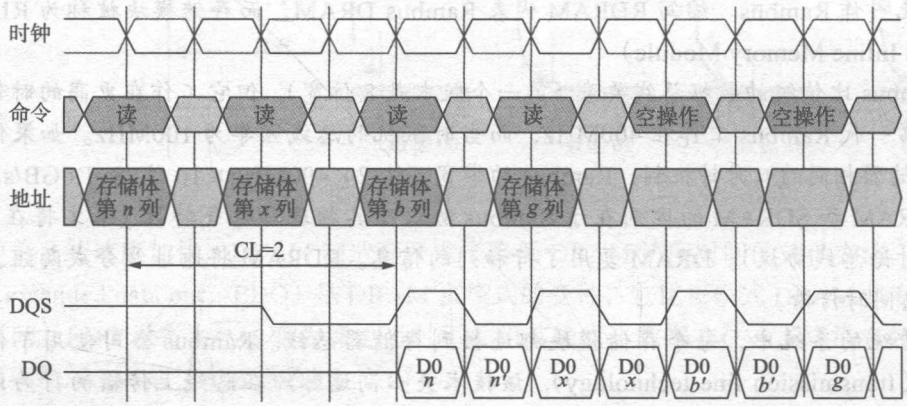
（续）

类型	位宽	通道数	时钟频率 /MHz	带宽 /MB/s
PC1066	16	单	533	2133
PC1200	16	单	600	2400
RIMM 3200	32	双	400	3200
RIMM 4200	32	双	533	4200
RIMM 4800	32	双	600	4800
RIMM 6400	32	双	800	6400

虽然 Rambus 推出时反响热烈，但它在 1999 出现以后就从来没有取得真正的成功。其接口的复杂性增加了制造成本，实际上，它与其他新形式的 DRAM 存储器相比并没有什么优势。最终，它被证明是计算机设计史上的另一个（昂贵）方案的脚注。

2. DDR DRAM

DDR DRAM 或者 double-data rate SDRAM，（双数据速率 SDRAM），与传统 SDRAM 的内部存储机制稍有不同。DDR DRAM、SDRAM 之间的区别在于其接口。DDR SDRAM 在时钟的上升沿和下降沿都执行数据访问；即它以时钟频率两倍的速度提供数据。图 2-28 给出了 DDR 的读周期时序[⊖]。一旦开始进行突发访问，数据在时钟的每个边缘都可出现。参数 CAS 延迟（CAS Latency，CL）以时钟周期为单位定义了从列地址选通有效至数据有效之间的时间。此参数是反映 DDR 性能的主要参数。其他 3 个与 DRAM 相关的参数是 RL（读延迟）、AL（附加延迟）和 BL（突发长度）。



Micron Technical Note TN-46-05,http://download.micron.com/pdf/technotes/TN4605.pdf.Courtesy of Micron Technology.

图 2-28 DDR DRAM 读周期

关于 DRAM 速度的术语

用 PC 来描述存储器标准有一些混乱。DDR SDRAM 被描述为 PC1600、PC2100、PC2700 等，它们定义了存储器的带宽。SDRAM 标准写为 PC66、PC100 和 PC133，描述了 SDRAM 的时钟频率。某时钟频率为 100MHz 的 DDR SDRAM 被称为 DDR200，而具有 64 位数据总线、可以同时传输 8 个字节的存储系统的带宽为 8×200=1600MB/s（因此命名为 PC1600）。

⊖ General DDR SDRAM functionality, Micron 技术说明 TN-46-05。

3. DDR2 与 DDR3 DRAM

每种技术的变种如 DDR DRAM 都有其寿命。DDR 风光后由 DDR2 代替。DDR2 风光后就引入了 DDR3。现在 DDR4 还在等着。在微机世界中，每种技术的变种都伴随着新的主板芯片系列为其提供连接到主机 CPU 的接口。

与 DDR 相比，DDR2 也是在时钟的上升沿和下降沿传输数据，但其更进一步，可以在每个时钟周期内执行 4 次数据传输。某个具有 64 位数据总线以 266MHz 频率工作的 DDR2 存储模块能够获得传输数据的峰值速率是 $64 \times 266 \times 10^6 \times 4/8$=8512MB/s。式中 4 表明每个周期传输 4 个数据。*数据传输速率*是数据值的 1/8，因为每次数据传输需要 8 位。DDR2 的最大信息传输速率是 1066 兆次传输 /s。

DDR2 和这个系列的后来成员在某种意义上说是可编程的，这是由于它们具有参数配置寄存器，可由主机系统装载操作参数。例如，可以定义 DDR2 SDRAM 如何进行刷新（前面说过，每个位元都需要周期性地更新，存储的数据需要再写一遍）。

DDR3 是数据传输速率加倍原理的进一步扩展，相比 DDR2 其数据传输率又翻了一番，可以在每个时钟周期内进行 8 次访问。在增加数据传输率的同时，DDR3 模块可以允许单个芯片的容量高达 8Gb。Samsung 在 2009 宣布第一款使用 50ns 技术的 4Gb DDR3 DRAM，这仅仅在 Intel 刚成立时发布其 1024 位的 DRAM 后的 40 年。那时的人们很难想象，存储芯片的容量将在他的一生中以超过 400 万倍的速度增长。DDR3 存储器内部由 8 个存储体而不像 DDR2 那样由 4 个存储体构成。增加存储体的个数可以增加交叉的程度。DDR3 内核速度与 DDR2 的稍有不同。

DDR3-1600 模块工作在 200MHz，其周期时间为 5ns，执行速度为 1066 兆次传输 /s，对应的数据传输率为 12800MB/s。图 2-29 给出了 DDR2 突发读操作的时序（来自 Samsung DDR2 SDRAM Device Operating & Timing Diagram manual）。

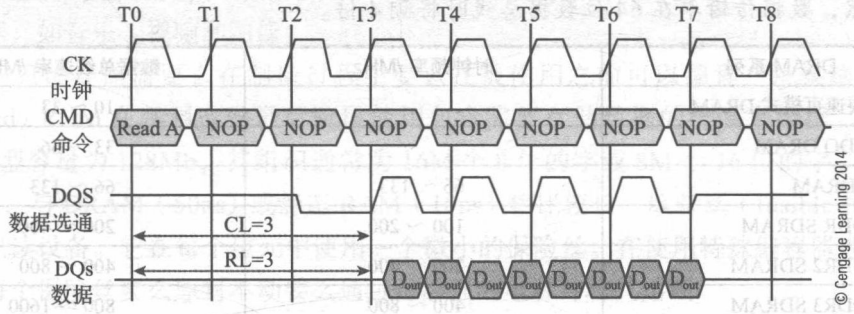


图 2-29　DDR2 DRAM 突发读模式

动态存储器设计的一个重要趋势是降低工作电压。DDR2 工作在 1.8V，第一代 DDR3 工作在 1.55V，后来的 DDR3 工作在 1.35V。工作电压从 1.8V 降至 1.35V 可以降低功耗 25%。

DRAM 技术通过向运行多任务的复杂操作系统以及处理音频和视频的多媒体应用提供大容量、经济的、直接访问的存储器，使得 PC 世界蓬勃发展。下面会介绍只读存储器，它虽然只对 PC 世界有适度的影响，但对普适计算（例如，MP3 播放器和数码相机）有更大的影响。

Fly-By 拓扑

主板上 DDR3 存储模块的布局或拓扑与 DDR2 存储器不同。该变化是时钟和信号速率增加的必然，这意味着早期计算机中不感兴趣的电路作用现在对计算机产生显著的影响。

在基于 DDR2 的系统中，信号并行交给所有模块。在基于 DDR3 的系统中，通过提供 Fly-By 拓扑来改进信号完整性；即该信号先被送到第一模块，然后是第二模块，等等。因此，信号依次飞过（fly by）每个模块。

这种新拓扑结构的影响之一是要求各模块的时序参数不同；即必须调整模块。在启动时，DRAM 控制器在被称为读写平衡（read and write leveling）的过程中调整各 DRAM 模块的参数。

4. DDR4

2011 年，正当 DDR3 以新的设计替换 DDR2 并越来越受欢迎的时候，DDR4 出现了。Samsung 是第一家提供 DDR4 模块的公司。新的 DDR4 技术比 DRAM 家族中以前的成员更快且能效更好。它采用新的电路设计技术，并且将电压从 DDR3 的 1.55V 降至 1.2V 以减少功耗。自 Intel 1103 DRAM 需要 3 股电源（+5V、−5V 和 12V）开始，已经走过了漫长的道路，该模块的数据传输率仅为 2133 比特 /s。

介绍完主流的读 / 写存储设备后，在讨论正在兴起的（目前已经出现）未来主存储器支撑技术之前，先介绍一下只读存储器。

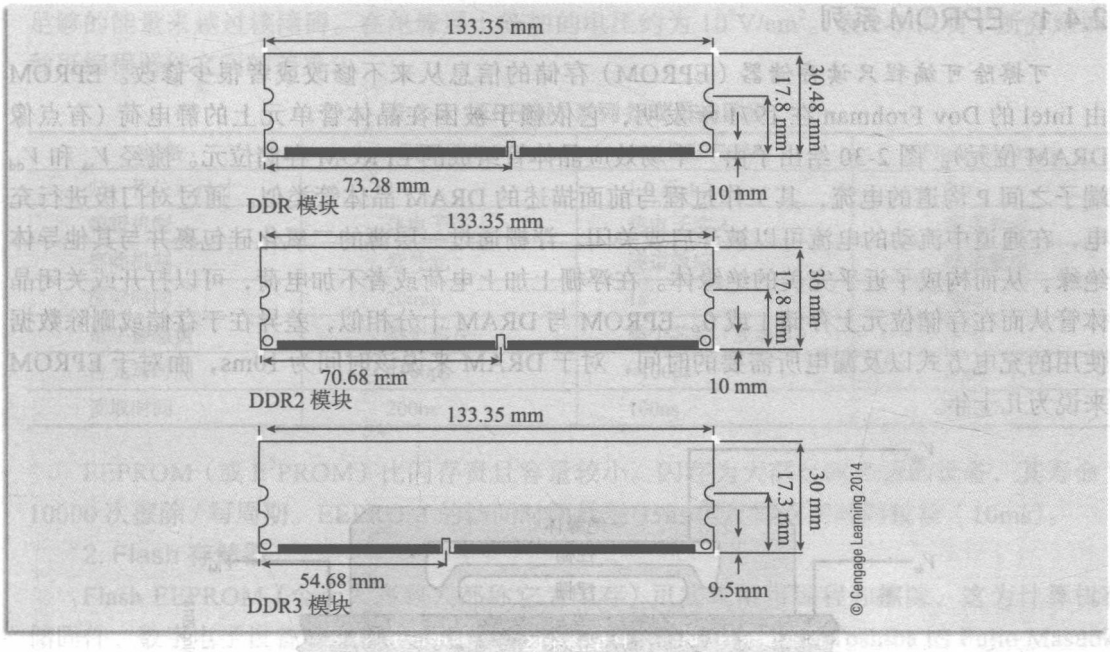
DRAM 的分代

下表显示了 DRAM 连续各代的进展。表中给出了 DRAM 的名称、时钟频率（除了基本的 DRAM 和 EDO RAM 没有统一时钟），假定数据总线为 16 位情况下的数据传输率。当然，数据传输率在 64 位数据总线时将翻 4 倍。

DRAM 系列	时钟频率 /MHz	数据总线速率 /MB/s
快速页模式 DRAM		10 ～ 33
EDO DRAM		33 ～ 66
SDRAM	66 ～ 133	66 ～ 133
DDR SDRAM	100 ～ 200	200 ～ 400
DDR2 SDRAM	200 ～ 400	400 ～ 800
DDR3 SDRAM	400 ～ 800	800 ～ 1600
DDR4 SDRAM	800 ～ 1600	1600 ～ 3200

DRAM 模块的外观

下图给出了三代 DRAM 模块的尺寸。每种模块都有一个在不同位置的关键点（缺口），以确保正确的模块被插入正确的主板。



2.4 只读存储器系列

只读存储器是内容可以访问但不能被修改的设备。今天，只读存储器（read-only memory，ROM）具有更广泛的意义，它也包括大部时间为读即经常读但不经常写的存储器（例如 PC 的 BIOS，它只在新的固件版本升级时才会改写）。在打开计算机之前，需要使用只读存储器来储存信息。例如，加载操作系统或其他系统参数的引导程序。只读存储器对于无盘系统来说至关重要（例如，手机，MP3 播放器，数码相机）。此时，ROM 用来保存程序和用户数据，如音乐、视频和图像。

真正的只读存储器要么在制造过程中要么在被使用之前可以编程。掩模编程（Mask Programmed）ROM 在其制造的最后阶段使用包含需要存储数据的掩膜来编程。掩模编程 ROM 的典型容量为 128Mb，其组织通常为 16M 个 8 位的字或 8M 个 16 位的字。其访问时间为 100ns，与 DRAM（50ns）或静态 RAM（10ns）相比较长。熔断丝（fusible link）ROM 是另一种只读设备。它在每个位元中使用一个微小的保险丝。在使用特殊编程器进行初始化编程时，每个保险丝要么原封不动要么通过足够强的电脉冲将其熔断。

掩模编程 ROM 仅适用于大规模生产且不需要重新编程的情况，熔断丝 ROM 仅用于特殊应用场合。成本低、容量大的应用，如电子词典和语法书用掩模编程 ROM 比较合适。计算机世界需要这样一种类型的只读存储器：它可以存储大量的数据，还可以花费较小的代价进行重新编程。

本节中，将介绍电可编程半导体只读设备家族中的 3 个成员，它们是 EPROM（第一个电可编程只读存储器）、EEPROM 和 Flash EPROM。这 3 种存储器十分类似，唯一的不同是对它们进行编程和擦除的特点与机制。然而，今天 Flash EPROM 已经成为只读（或大部时间为读）存储器的主导形式。

2.4.1 EPROM 系列

可擦除可编程只读存储器（EPROM）存储的信息从来不修改或者很少修改。EPROM 由 Intel 的 Dov Frohman 在 1971 年发明，它依赖于被困在晶体管单元上的静电荷（有点像 DRAM 位元）。图 2-30 给出了由一个场效应晶体管组成的 EPROM 存储位元。流经 V_{ss} 和 V_{dd} 端子之间 P 沟道的电流，其工作过程与前面描述的 DRAM 晶体管类似。通过对门极进行充电，在通道中流动的电流可以被开启或关闭。浮栅通过一层薄的二氧化硅包裹并与其他导体绝缘，从而构成了近乎完美的绝缘体。在浮栅上加上电荷或者不加电荷，可以打开或关闭晶体管从而在存储位元上存储 1 或 0。EPROM 与 DRAM 十分相似，差异在于存储或删除数据使用的充电方式以及漏电所需要的时间，对于 DRAM 来说该时间为 10ms，而对于 EPROM 来说为几十年。

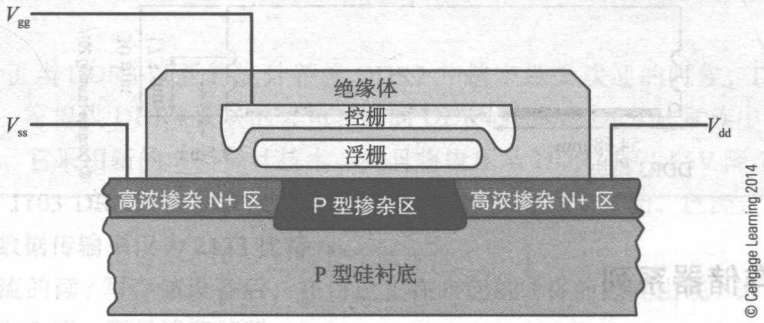


图 2-30 EPROM 存储位元

如何对完全绝缘的浮栅进行充电？浮栅附近有一个控栅，它也与浮栅绝缘。通过向控栅施加高电压（例如，12 ～ 25V），电子将穿过绝缘体到达浮栅。虽然 12V 看起来不像一个高电压，但电位差（potential gradient）将产生足够大的力使电子通过只有很小间隙的绝缘体。

一旦对 EPROM 编程，数据将保留在浮栅上 10 年或更长时间。如果需要消除电荷，必须将芯片放在紫外灯下并打开芯片表面使其暴露在紫外线中。因为玻璃对紫外光来说是不透明的，故 EPROM 芯片的背面装有用石英制成的透明窗户。此时所有数据将被立即删除。

第一代 EPROM 的容量小，编程需要轮流对各个位置写数据。第二代大容量 EPROM 使用智能编程算法（smart programming algorithm）向每个存储位元发出短编程脉冲，且重复操作至所有数据均已被正确地写入。有些 EPROM 是一次性可编程的（one-time programmable），因为它们没有石英窗口（因为代价较高），一旦编程就保持不变了。

EPROM 提供了在实验室中开发计算机固件的一种手段。不幸的是，其缓慢的编程过程和擦除的繁琐机制使得 EPROM 不适合那些甚至只是偶尔需要编程的应用。今天，紫外线可擦除的 EPROM 已经基本过时，剩下的 EPROM 只是在 eBay 网上等待死亡。

1. EEPROM

电可擦除可编程只读存储器（EEPROM）建立了原始的 EPROM 和今天的闪存之间的联系。本节通过展示 EPROM 的发展来介绍 EEPROM。EEPROM 和 Flash EEPROM 之间的主要区别在于数据的擦除方式。EEPROM 由 Intel 的 George Perlegos 在 1978 年研发。

在 EPROM 中，绝缘体中的被困电子通过紫外光光子被删除。在 EEPROM 中，绝缘层很薄以至于当芯片被擦除时一种被称为 Fowler-Nordheim 隧道的量子力学效应可以使电子穿过绝缘层。当在绝缘层施加一定的电压时，浮栅上的电子就可以通过绝缘层，即使它们没有

足够的能量来越过该障碍。在绝缘层上施加的电压约为 10^7V/cm^2。表 2-3 说明了所介绍的 3 种可编程器件之间的差异。

表 2-3 EPROM 系列各成员的区别

设备	EPROM	EEPROM	Flash Memory
归一化位元尺寸	1.0	1.0 至 1.2	3.0
编程机制	热电子注入	热电子注入	隧道效应
擦除机制	紫外光	隧道效应	隧道效应
擦除时间	20min	1s	5ms
最小擦除量	整个芯片	整个芯片（或扇区）	字节
位元写时间	<100μs	<100μs	5ms
读取时间	200ns	100ns	35ns

EEPROM（或 E^{2}PROM）比闪存贵且容量较小。闪存为大部时间为读的设备，其寿命为 10000 次擦除 / 写周期。EEPROM 的访问时间低至 35ns 但其写周期时间较长（10ms）。

2. Flash 存储器

Flash EEPROM（今天大多数人都称它为闪存）可实现电可编程和擦除，这为计算机存储固件、数字电子设备和便携式应用提供了一种方便的手段。它由 Toshiba 的 Fujio Masuoka 在 1980 年发明。图 2-31 给出闪存位元的结构。EPROM 中介于浮栅 MOS 晶体管表面之间的氮氧化硅绝缘层（ONO）的厚度约为 300Å，而 Flash EEPROM 只有 100Å（1Å=1 × 10^{-9}m 或 1nm）。人类眼睛可以看到的可见光的波长为 390 ～ 750nm，这意味着浮栅的厚度约为最高频率可见光（蓝色）的一半。

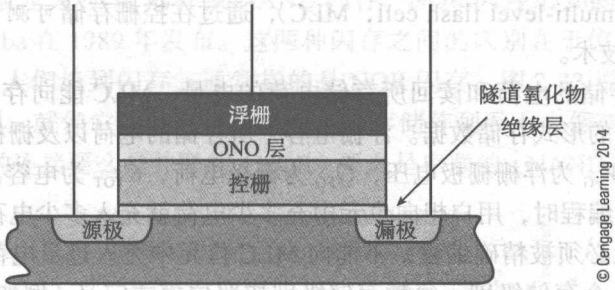


图 2-31 Flash EEPROM 存储位元

对 EPROM 编程时，可以通过雪崩效应（avalanche effect）让电子穿过氮氧化物绝缘层来对浮栅充电。这些电子由于其高动能被称为热电子（hot electrons）。Flash EEPROM 通过穿过绝缘体的电子隧道进行编程并以同样的方式进行擦除。表 2-3 比较了 3 种可编程设备的写 / 擦除机制。

用户无法擦除 Flash EEPROM 中某个位元中的数据。Flash EEPROM 通常被分为典型大小为 1024 字节的扇区。某些设备允许用户擦除某个扇区。第一代 Flash EEPROM 只能执行 100 次擦除 / 写周期，而现在的设备通常的寿命为 10000 次擦除 / 写周期。实际的 Flash EEPROM 的寿命可能还要大一个数量级。

Flash EEPROM 的编程方式与 EPROM 一样。它们与系统的读接口与 SRAM 的读接口相似，具有低电平有效的写使能输入信号。其写周期与传统 SRAM 的相同，只是写持续时间要长很多。Flash EEPROM 具有片上定时器和相关的控制电路，可以自动保证适当的信号延

迟而不需要使用外部硬件。某些 Flash EEPROM 一次可以对一个字节编程，而另一些则可以在一次操作中对整个扇区（例如，1024 字节）进行写。Flash EEPROM 的擦除机制因制造商而各不相同。它们可以一次全部擦除也可以一次擦除一个扇区。

闪存技术

现代普适计算的两大支柱是 USB 总线和闪存。USB 总线（将在第 4 章讨论）允许用户将各种数字系统（数码相机与电脑，手机与 iPad 等）连接到一起。数据传输率和信息交换协议被自动处理，对用户透明。同样，闪存已经从 8MB 的设备发展到 2010 年容量为 512GB 的闪存卡。有趣的是，用户可以购买一块 256GB 的闪存驱动器，它是 USB 和闪存技术的结合，可使用户掌握便携的 256GB 的存储系统（可保存一辈子中的文档和程序，若要保存图像和多媒体则需要更多的存储空间）。

闪存技术的不断进步包括 2010 年在笔记本中出现的固态硬盘。将笔记本电脑中的硬盘更新为闪存提高了性能（包括数据传输和访问时间）、降低了功耗、提升了可靠性（没有脆弱的运动部件）。我们将在下一章详细介绍硬盘。

EPROM 通常为 32Mb，被安排为 4MB。现代闪存可以是容量为 16Gb（2G×8）的 NAND 型也可以是 256Mb（16M×16）的 NOR 型。NAND 和 NOR 型闪存位元的内部组织不同，后面还将介绍闪存这方面的内容。

3. 多级闪存技术

大家知道，DRAM 和 EPROM 系列，都是通过电容中的电荷来控制通道是否导通来存储数据的。这些存储技术唯一的差别是它们的结构和操作参数。20 世纪 90 年代末，Intel 开发了多级闪存位元（multi-level flash cell，MLC），通过在控栅存储可测量的电荷（measured charge）来扩展闪存技术。

通过仔细控制存储的电荷和读回所存储电荷的电量，MLC 能向存储位元中写入多位；即多级技术以模拟量的形式存储数据。浮栅电容、所存储的电荷以及栅极电压之间的关系是 $V_{FG}=Q_{FG}/C_{TOT}$，其中 V_{FG} 为浮栅栅极电压，Q_{FG} 为栅极电荷，C_{TOT} 为电容。

当对传统的闪存编程时，用户想向位元中充多少电荷就充入多少电荷，只要它足以改变状态即可。多级位元必须被精确编程；不能向 MLC 位元中充入过量电荷，否则就会把某个存储级别转换成另一个存储级别。每种存储级别按照位模式定义（例如，00,01,10,11）。传统的闪存或 SLC 闪存感知栅极电压，并将其与阈值比较，然后根据结果在阈值的上方还是下方分配一个 1 或 0。MLC 技术将栅极电压与几个参考电平进行比较，并为每个级别分配一个二进制代码。每存储 10000 个电子浮栅上的电压大约增加 1V[⊖]。图 2-32 显示了 SLC 和 MLC 位元的状态分布。

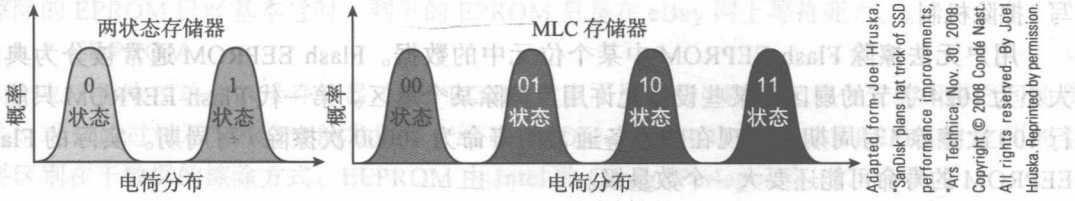


图 2-32 SLC 闪存与 MLC 闪存的电荷分布

⊖ Al Fazio and Mark Bauer, "Intel StrataFlash Memory Development and Implementation," *Intel Technology Journal*, Q4'97.

晶体管老化

具有运动部件的机械装置，如发动机或磁盘的磁头，经过一段时间会磨损。一般的观点认为，如集成电路这样的设备因为使用晶体管所以不会磨损或退化。但这是不对的。

2011 年年初，许多人惊讶地发现在为第二代 Core i5 和 i7 处理器设计的支持芯片中存在缺陷。该问题只影响经过一段时间的运行后（估计为 3 年）3Gb/s 的 SATA 硬盘接口。故障被追溯到一只电压太高的晶体管，它会随时间退化。

Sandy Bridge 问题给人们一个提醒，晶体管不是钻石，它不能永远正常工作。晶体管退化有几个原因。随着时间的推移，电子从传导通路中漂移出去并被困在介电绝缘层中，影响了开关的阈值。介电层可能随着时间的推移，由于电压而破裂。也有可能是用来连接芯片的铜或铝原子逐渐扩散到硅中并改变了其属性。

由于将电子从存储数据的控栅移进移出所需的静电场很大，闪存更容易失效。

具有 4 个参考水平的位元可以存放两位数据；有 8 个参考水平的位元可以存放 3 位数据；等等。可以保存的级别数量只取决于存储可精确测量的电荷的能力以及将栅极上的电压与参考电压进行精确比较的能力。给定位密度，MLC 可以减少有效位元面积和单芯片的尺寸。这最终会导致每兆字节成本的显著降低。今天，MLC 存储产品每个存储位元可以存放两位数据。

当前的 MLC 存储器被认为是一种消费品而非工业产品，这是因为它不能工作在工业温度范围内，也不如 SLC 闪存那样可靠。MLC 最大写次数通常是 SLC 闪存的 10%。

4. NAND 与 NOR 闪存

现代的闪存被描述为 NOR 闪存或 NAND 闪存。NOR 闪存由 Intel 在 1988 年发布，而 NAND 闪存由 Toshiba 在 1989 年发布。这两种闪存之间的区别在于位元的组织（即连接方式）。一般来说，当人们谈到闪存，通常指的是 NOR 闪存。图 2-33 说明了 NOR 和 NAND 结构[⊖]最基本的区别。就像它们的名字所显示的，存储阵列看上去像或非门 NOR 或者与非门 NAND，其单元的连接要么是并联的（NOR）要么是串联的（NAND）。

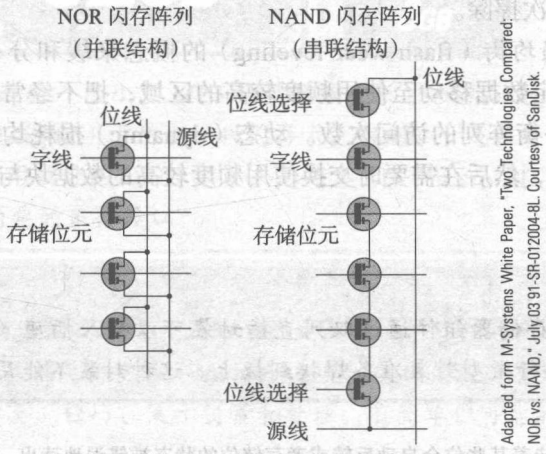


图 2-33 NOR 和 NAND 闪存技术

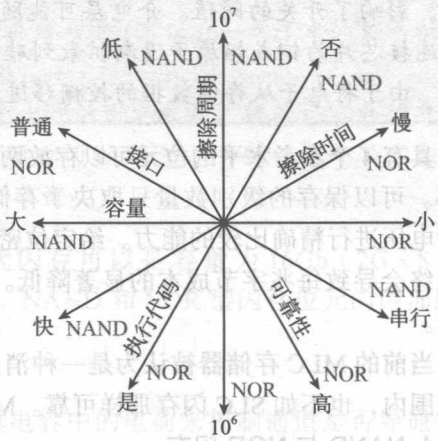
⊖ M-Systems 白皮书，"Two Technologies Compared: NOR vs. NAND." July 03 91-SR-012004-8L.

这两种技术之间的主要差异在于它们的操作特性（operating characteristics），因此也表现在它们的应用领域。NOR 闪存的优点是代码可以直接在其中执行，称为芯片内执行（eXecute In Place，XIP）原理。代码不能在 NAND 闪存内执行，必须先传输给静态存储器或 DRAM。NOR 通常用来实现相对较小的存储器，如 4MB 的存储器。

NAND 闪存可以提供更高的位元密度，但需要更复杂的系统接口（因此不能直接运行代码）。NOR 闪存的擦除块可达 128KB，需要 5s 来擦除；而 NAND 擦除块可能是 32KB，只需要 4ms 来擦除。NOR 闪存的系统接口同 SRAM 的十分相似，而 NAND 闪存具有串行的一位的接口，并不是所有的供应商都实现了相同的串行数据传输协议。

NAND 闪存的另一个优势是其擦写次数可达 100 万次，远远高于 NOR 闪存的 10 万次。虽然允许的擦除次数较多，但 NAND 删除比 NOR 闪存的可靠性差，需要使用纠错码来处理 1 位错误[⊖]（类似在下一章中讨论的 CD-ROM 和 DVD）。

NAND 闪存中包含比所需更多的数据块（即它的存储容量大于标称值）。这种冗余是必要的，因为某些块可能是坏块（bad blocks）。系统软件持续监控存储器操作，并在碰到坏块时将其替换（在硬盘驱动器中也会有类似的过程）。来自 Toshiba 报告中的图 2-34 给出了 NAND 与 NOR 技术的相对差异[⊖]。



Adapted form M-Systems' White Paper, "Two Technologies Compared: NOR vs. NAND," July 03 91-SR-012004-8L. Courtesy of SanDisk.

图 2-34 NOR 和 NAND 闪存的比较

5. 闪存的损耗均衡（Wear Leveling）

由于电子被注入位元或从位元中删除时绝缘体可能会受损，故闪存位元具有有限的寿命。如果对闪存阵列中所有位元进行相同次数的访问和删除操作，阵列中所有部分的老化程度相同。然而，实际情况是，对存储位元的访问并不均匀。例如，在 MP3 播放器中，某些音乐可能没被听过，而其他音轨被经常播放。因此，数组中某些位元可能被永久损耗，而其他位元仍然能够进行多次擦除。

可以采用闪存耗损均衡（flash wear leveling）的概念来缓和分布不均匀地擦除。静态（static）损耗均衡将固定数据移动至使用频度较高的区域，把不经常使用的区域分配给经常变化的数据，即试图平衡阵列的访问次数。动态（dynamic）损耗均衡算法更复杂，它先监控闪存阵列的使用情况，然后在需要时交换使用频度较高的数据块与使用频度较低的数据块的位置。

CF 卡

传统上，半导体存储器组件通过双列直插封装可以插入插座，或被自动机械直接焊接到 PCB 板上，或进行微型封装准备焊接到板上。这种封装不能用于具有互操作性的消

⊖ 闪存的自然属性意味着某些位会自动反转或者存储位的状态被错误地读出，此种情况称为比特翻转（bit-flipping）。该问题在 NAND 闪存中比在 NOR 闪存中更常见，这使得 NAND 闪存更适合多媒体业务，因为偶发的位错误并不会产生重要影响。

⊖ TOSHIBA, "NAND vs. NOR Flash Memory Technology Overview." Toshiba America Electronic Components, Inc. Irvine. CA.

费者应用中。1994 年，出现了 CF（Compact Flash）存储卡，可以使用户能够将存储器插入家用基于个人计算机的系统。这些系统中最重要的代表是数码相机和 MP3 播放器。

CompactFlash 协会（CFA）于 1995 年 10 月成立，以促进 CompactFlash（CF）技术，并与供应商一起规范 CF 技术。

CF 存储器是一个相对较小的部件，尺寸为 43mm×36mm（1.7 吋 ×1.4 吋）且只有 3.3mm（0.13 吋）厚。其大小约为在第 4 章中介绍的 PC 卡的一半。事实上，用户可以为 CF 卡购买非常便宜的 PC 卡适配器，方便将数码相机中的照片复制到笔记本中。前面描述的是 CF 卡 I 型，还有 5mm 厚的 II 型。

CF 模块的优点是可靠性高、不易受冲击、低成本。它们被设计成可以支持双电压标准（5V 和 3.3V），这是非常明智的，这使得 CF 模块与现代的以及旧的系统都可以兼容。

除了实现闪存外，CF 卡的格式也针对其他功能（例如，微型硬盘和如以太网和蓝牙无线适配器这样的 I/O 设备）进行了调整。提供诸如 I/O 这样的扩展功能的 CF 卡被称为 CF+ 卡。

CF 卡与主机系统的接口使用 50 针连接，与 PC 卡连接器类似。CF 卡到外部系统的接口并不是简单的存储接口（即它由数据、地址和控制线构成）。CF 接口提供了总线接口，这意味着卡内可以实现一定程度的智能。

CF 卡智能控制器管理接口协议、数据存储和检索、纠错码以及诊断。接口使用配置寄存器和软件管理系统。从主机系统的角度来看，CF 卡看起来像一个标准的 ATA 磁盘接口（即 IDE）。

LIU, CHIN-CHENG/Shutterstock

LIU, CHIN-CHENG/Shutterstock

CF 卡的尺寸与今天的一些小型 MP3 播放器和傻瓜相机相比较大，所以在 1999 年出现了称为 SD（secure digital）卡的新一代闪存卡。SDHC 卡是容量大于 2GB 的 SD 卡。相应地，为诸如手机应用设计的比 SD 卡更小的微型 SD（micro-SD）卡在 2005 年出现。SD 卡比 CF 卡具有更简单的串行接口。

记忆棒

Sony 是一家在时尚消费产品设计领域领先的公司，在 1988 年创建了自己的可交换存储器的标准，该存储器就是记忆棒（Memory Stick）。该设备具有针对消费者市场产品的所有属性。它结构紧凑、轻巧、便于携带和处理，有简单但可靠的电气接口，可以为保护数据而被设为只读。

第一代容量高达 64MB 的记忆棒在 1998 年推出。2002 年，最大容量已增加到 512MB，接口升级到 20Mb/s。目前的记忆棒可提供 32GB 的容量，升级的记忆棒（Memory Stick Pro）的容量高达 2TB。然而，Sony 在 2010 年宣布支持 SDHC 卡为记忆

棒的长远未来投下了阴影。

为兼容基于PC的系统，记忆棒采纳了PC FAT文件管理系统。然而，Sony设计的记忆棒不仅用于在应用之间传输数据，还为了控制受版权保护的数据流动。音乐和电影产业公司极大地受益于互联网和新技术，因为它们有助于扩大市场。同样，音乐产业对媒体的作用不满意，因为大规模互联互通与数字技术使音频和视频文件可以在没有质量损失的情况下任意复制。Sony采用MagicGate技术与记忆棒配合，以禁止未经授权的数据存储。MagicGate版权保护包括两种技术：确定设备是否合法地支持MagicGate的身份验证（authentication）技术，以及保护合法内容的加密（encryption）技术。每个记忆棒都具有用MagicGate生成的独一无二的ID号。

与闪存卡的并行接口不同，记忆棒在卡与主机之间具有串行接口。串行接口的优点是简单。记忆棒只提供10个接口引脚，包括为未来扩展的引脚。这么简单的接口使得它更容易具有轻薄的外形。这对小型个人设备来说是重要的，可以确保接口的机械可靠性且不会出现由于频繁拔插导致的断续接触问题。

记忆棒由Flash EPROM构成，由串行接口控制器管理存储，并具有MagicGate版权保护机制。该控制器提供了闪存的接口，可以进行修改以适应不断出现的新存储技术。控制器执行串行到并行的转换（反之亦然）来处理错误检测和校正。实现串行接口的3个信号为：

SCLK　　（串行时钟）

SDIO　　（流入和流出存储器的常见双向串行数据路径）

BS　　（突发状态信号）

主机总是通过SCLK和BS来初始化与记忆棒的通信。串行数据信号SDIO传输包含CRC错误检测机制的512字节的数据帧。突发状态信号表示数据传输开始。BS也用来区分RDY/BSY消息和中断消息。突发状态信号将SDIO双向数据线上的数据分为4种类型：

状态模式	BS电平	操作
BS0	低	中断——没有数据传输
BS1	高	传输控制协议状态——从主机拷贝TCP到记忆棒
BS2	低	写协议的数据传输状态；将数据传输到记忆棒。读协议的握手状态。等待RDY信号就绪
BS3	高	写协议的握手状态；等待RDY信号就绪。读协议的数据传输状态。从记忆棒读取数据

在状态BS1对传输协议控制命令进行传输，在BS2和BS3状态，定义传输数据类型和正确的协议。然后，SDIO上的信号传输数据并基于协议完成握手。

记忆棒在诸如数码相机、录音机及MP3播放器这样的音视频设备中使用。由于信息表示静止的或移动的图像和声音/音乐，大多数数据传输量会比较大。如果交换的数据量小，对数据传输进行优化是无意义的，因此最小的数据交换量被设置为8KB。数据使用文件分配表（FAT）机制以PC兼容的格式存储。然而，Sony已经确定了几个重要的影音文件格式，预定义了与这些应用兼容的目录管理机制。

2.5 新兴的非易失性技术

本节将介绍一些非易失性存储系统的新兴技术。某些技术还处于早期研究阶段，有些

技术已经制成产品。首先将介绍两种成熟的技术：铁电（ferroelectric）RAM和奥氏双向（Ovonic）存储器。

铁电RAM又称FRAM，是一种半导体非易失性存储器的形式，在20世纪90年代开始出现。虽然单词“ferroelectric”的前缀“ferro”通常表示与铁磁相关，实际上FRAM有点名不符实，它与磁技术无关。DRAM的原理是向某种材料充电从而保存电场。磁盘和磁带系统是在记录介质上记录磁化方向的连续排列从而达到保存磁场的目的。FRAM通过在晶格中移动某个原子来改变极化（polarization）从而实现数据保存。

铁电效应描述了材料在没有电场的情况下保存极化的能力，使用该术语的原因是因为其与磁性材料能够在没有磁场的环境下保存磁场的能力类似。

为了理解铁电效应，必须首先理解材料的几个简单属性。材料可以分为两类：导体和绝缘体。它们之间的区别是电子可以在导体中自由移动而不能在绝缘体中移动。如果向导体（例如铜）施加电场，则导体中的电子将在电场的影响下移动。如果将绝缘体放入电场，电子不能在绝缘体中移动。然而，电场确实对组成绝缘体的分子产生了影响，它扰乱了绝缘体分子结构中正负电荷的均匀分布。这种在电场的影响下导致电子和原子核位置发生改变的情况被称为极化。

图2-35和图2-36显示了电场对单个原子的影响。图2-35中，带有负电荷的电子围绕带有正电荷的原子核旋转，原子电荷的平均分布为零，这是由于正负电荷相互抵消。图2-36中，施加的电场使得对称性被改变。围绕原子核的电子云（electron cloud）的平均位置的改变量由电场的强度决定，此时原子获得了净电荷（net electric charge）。图2-37显示了原子可以被看成一个双极子（dipole），它有两个电极（这也是极化这个术语的由来）。双极子的电行为可以看成磁性的相似体。

图2-35 单个原子的结构

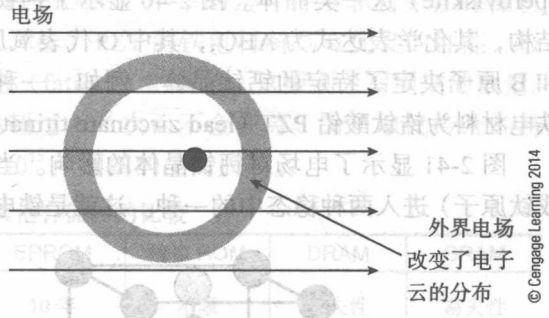


图2-36 电场对单个原子的作用

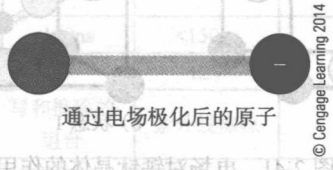


图2-37 电场中的原子表现出极化

正如磁性表现为从北至南排列那样，双极子也表现为从正极到负极的排列，如图2-38所示。一串双极子可以看成一个具有更强能量的双极子，如图2-39所示。在电场中的绝缘体产生了双极子，它就被称为电介质（dielectric）。绝缘体极化的强度用电介常数（dielectric

constant）来表示，可以用真空中的电介常数来理解。从本书的观点来看，极化材料最重要的效果是其可以用来保存数据。

图 2-38 双极子的排列

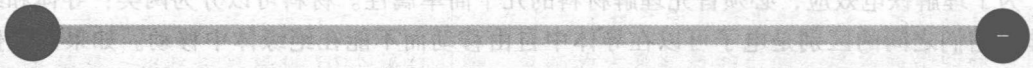


图 2-39 双极子排列的体效应（bulk effect）

FRAM 是铁电效应的实际应用。通过在微小的电容中使用铁电薄膜来保存数据。铁电薄膜根据被施加电场的方向而被极化为两个方向之一。

铁电技术在 20 世纪 60 年代早期由 Stanford 大学首先提出，S.Y.Wu 等人于 1974 年研究了铁电材料和半导体技术如何结合起来使用。1988 年，Ramtron 国际公司宣布了第一款商用的铁电随机访问存储器（FRAM）。

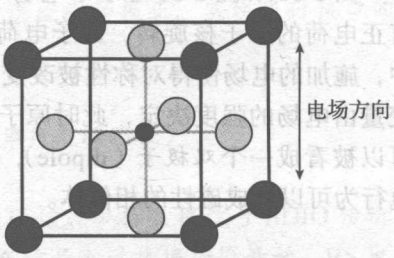
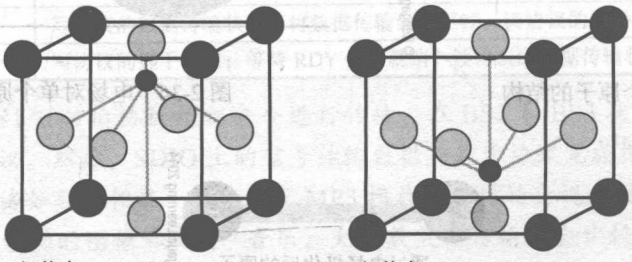


图 2-40 钙钛晶体的结构

在 FRAM 中使用的铁电材料属于为钙钛结构（peroviskite）这一类晶体。图 2-40 显示了钙钛晶体的结构，其化学表达式为 ABO_3，其中 O 代表氧原子，A 和 B 原子决定了特定的钙钛晶体。例如，一种常见的铁电材料为锆钛酸铅 PZT（lead zirconate titinate），它是 $PbZrO_3$ 和 $PbTiO_3$ 的混合物。

图 2-41 显示了电场对钙钛晶体的影响。当施加电场时，晶体中心的原子（该例中为锆或钛原子）进入两种稳态中的一种。这就是铁电钙钛晶体保存数据的机制。



a）状态 0　　b）状态 1

图 2-41 电场对钙钛晶体的作用

2.5.1 铁电迟滞

考虑在两个金属板间夹着一片铁电材料的系统，如图 2-42a 所示。假设在电介质两端施加电压使之极化，如图 2-42b 所示。移除电压后，对电介质的极化不会产生影响，如图 2-42b

所示。同样，在金属板间施加一个小的无论是哪个方向上的电压，都不会产生整体影响。

如果在板间的电压逆转，并增加其强度，如图 2-42c 所示。电介质的极化方向将发生改变，材料的状态发生变化。当铁电材料中的原子从晶体的一端移动到另一端以至于被检测到时，则出现脉冲电流。人们无法检测到这些设备到底处于哪种状态或者极化方向，但可以检测其是否改变了状态，即 FRAM 存储器具有破坏性读出。

图 2-42d 中，电源再次被移除，铁电材料保持其中的电荷。此时已拥有存储器需要的一切：迫使材料进入两种状态之一的方法，以及通过在两端施加电压来检查其处于哪种状态的方法。

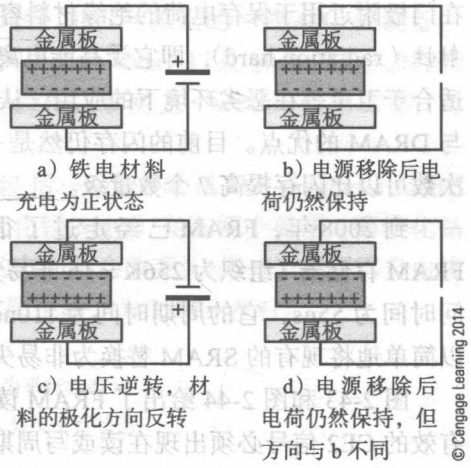


图 2-42 铁电存储

实用的 FRAM 位元使用稍加修改的 DRAM 技术构建。刚才介绍了通过向铁电材料充电可以实现将数据写入 FRAM 位元。为了读取存储位元中的数据，选中该晶体管，然后施加电场穿过电容。如果位元已经被极化为与电场一致的方向，则什么都不会发生。然而，如果位元是反向充电的，则开关晶体管中有电流流过。

FRAM 在约 100ns 的时间内改变状态，这比一些传统的半导体非易失性存储器（如 EPROM）的速度要快，但比 SRAM 或 DRAM 的要慢。读 FRAM 位元的动作当然是破坏性的，这是由于数据是通过位元改变了、还是没有改变来检测的。因此，铁电存储器读周期后往往接着一个写周期来写回数据，原来保存的数据可能改变也可能没有改变。

表 2-4 比较了半导体非易失性存储器技术和 FRAM。不幸的是，许多非易失性存储机制，包括 FRAM，其改变状态的次数往往受限。基于 PZT 的铁电材料具有优良的性能，极化程度高，易于批量生产。但它具有被称为*疲劳*（fatigue）的现象，这限制了写周期的数量。较新的材料，如 SBC（即 $SrBi_2Ta_2O_9$），可极化超过 10^{12} 次而不会疲劳。为了取代传统的易失性 RAM（如 FRAM 这样的设备），需要具有 10^{15} 个周期的生命周期。

表 2-4 FRAM 与其他存储器的比较

	FRAM	EEPROM	闪存	EPROM	掩模 ROM	DRAM	SRAM
数据保存时间	10 年	10 年	10 年	10 年	不限	易失性	易失性
存储位元结构	1 个晶体管 +1 个电容	2 个晶体管	1 个晶体管	1 个晶体管	1 个晶体管	1 个晶体管 +1 个电容	6 个晶体管
读时间	180ns	200ns	<120ns	<150ns	<120ns	70ns	70 ～ 85ns
写电压	2~5V	14V	9V	12V	—	3.3V	3.3V
重写方法	覆盖	擦除或改写	写和擦除的组合	紫外线擦除	—	覆盖	覆盖
重写周期	180ns	每字节 10ns	每扇区 1s	每字节 0.5ms	—	70ns	70 ～ 85ns
消除数据	不必要	必要（字节擦除）	必要（扇区擦除）	必要（紫外线擦除）	—	不必要	不必要
写周期数	$>10^{12}$	10^5	10^5	100		不限	不限
保持电流	20μA	20μA	5μA	100μA	30μA	1000μA	7μA

FRAM 存储器不需要承受 EPROM 中常见的高水平电应力使电荷通过绝缘体。EPROM 在门极附近用于保存电荷的绝缘材料容易出现灾难性故障。FRAM 的另一个优点是它的耐辐射性（radiation hard），即它受高能电离辐射的影响比某些半导体器件的影响要小，因此其更适合于卫星等在恶劣环境下的应用。从用户的角度看，FRAM 的实际优点是，它结合了闪存与 DRAM 的优点。目前的闪存仍然是一种大部时间为读的存储器技术，而 FRAM 的写操作次数可以比闪存提高 7 个数量级。

到 2008 年，FRAM 已经走过了很长的路，Ramtron 国际公司因其 FM22L16 4Mb 的 FRAM 存储器（组织为 256K × 16 非易失性 RAM）而荣获中国电子产品年度产品奖。它的访问时间为 55ns，它的周期时间为 110ns。其电气特性与行业标准的 SRAM 兼容。这样，可以简单地将现有的 SRAM 替换为非易失性 FRAM 而不需要修改硬件。

图 2-43 和图 2-44 给出了 FRAM 读写周期的时序图。低电平有效的 CE1 信号和高电平有效的 CE2 信号必须出现在读或写周期开始。19 位地址从 A_{18} ～ A_{00} 输入，相关输入 / 输出信号由 DQ_{15} ～ DQ_{00} 给出。这些图没有什么特殊之处，即它们与其他半导体 SRAM 的读写周期非常相似。

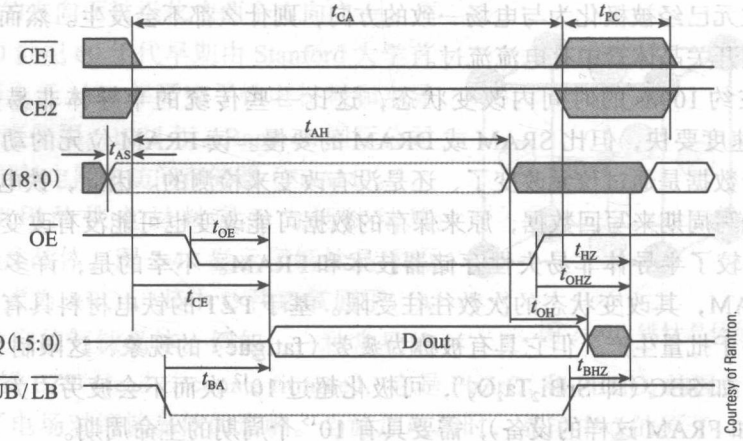


图 2-43 FRAM 读周期时序图（RAMTRON FM23MLD16 512K × 16）

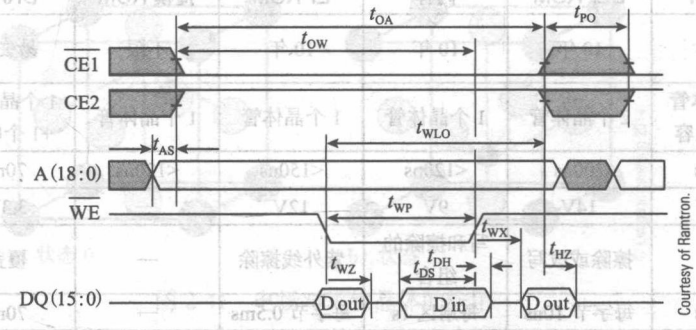


图 2-44 FRAM 写周期时序图（RAMTRON FM23MLD16 512K × 16）

2.5.2 MRAM——磁阻随机访问存储器

磁阻随机访问存储器（Magnetoresistive RAM，MRAM）利用了物质的电磁特性。就像

其他某些现代技术那样，其未来尚不确定，尤其是因为它正与其他的非易失性 SRAM 构成竞争。第一款商业化 MRAM 存储器是由 Freescale 公司在 2006 年推出的一种 4Mb 的设备。

MRAM 具有与半导体静态存储器类似的访问时间（5 ～ 40ns）以及 10ns 的写入时间。与闪存不同，MRAM 位元可以不受次数限制地进行写操作。事实上，MRAM 具有许多（与其竞争技术相比）近乎完美的存储器的特性。

图 2-45 展示了 MRAM 存储器元件的组成，包括一个磁隧道结（magnetic tunnel junction，MTJ）结构。MTJ 由 3 层组成：氧化层（氧化镁 MgO）夹在两个磁层（硼铁化钴 CoFeB）之间。其中一个磁层是固定的（即内部磁场的方向不会改变），另一个磁层可自由旋转其磁化方向，如图 2-45 上层中的双箭头所示。氧化层很薄，为 1.2nm 左右。

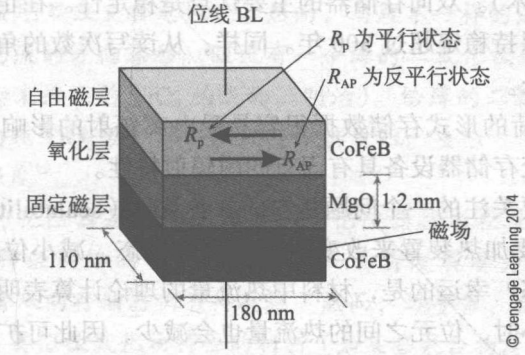


图 2-45 MRAM 单元

自由层的磁化方向可以在与底层平行和反平行之间进行切换。切换自由磁层磁化方向执行的操作是通过使电流通过靠近 MRAM 位元的写线来完成。线路中电流生成磁层所需的磁场。当两层的磁化方向相同，位元具有低电阻（即沿图 2-44 中位线 BL 的方向），当磁化是反平行的，其电阻较高。可以很容易地测量位元中的电阻，并可以用其代表二进制状态。此外，存储器位元不像 FRAM 那样是破坏性读出的。在下一章中，我们将利用该技术讨论硬盘驱动器的磁头。然而，截至 2011 年年底，MRAM 技术才开始形成商业化产品。以其高速、有效而不限次数的读 / 写周期，MRAM 宣称提供适合所有需求的通用存储器部件（除了高速 Cache）。

2.5.3 双向存储器

材料的另一种可以用来存储信息的特性是相（phase）。双向（Ovonic）存储器利用薄膜材料的相变来存储数据。该相变是可逆的，因此存储位元可被擦除和改写。双向材料要么处于结构化的晶态（crystalline），要么处于无组织（amorphous）非结构化的非晶态。具有适宜相变特性的一类材料是硫系（chalcogenide）玻璃，它由 Bell 实验室的 Ovshinsky 在 1968 年首次发现。

如果将硫系玻璃熔化并快速冷却，它将进入非晶相（amorphous phase）。如果它慢慢地被加热到略低于其熔点的温度，非晶态材料将恢复到其初始的晶相（crystalline phase）。非晶材料的特点是原子短程有序，具有高反射率和高电阻；而多晶材料的特点是原子长程有序，具有低反射率和低电阻。

由于双向材料处于两种状态时其电阻有四十倍的差异，因此可以用来记录数据。在下一

章中，将介绍相变机制也可以被用于光存储系统，例如可重写的CD和DVD。因为硫属化物可以制成薄膜，它可以结合在半导体单元中，使得双向存储器可以像其他的半导体存储器那样构建。

与闪存不同，相变存储器可以按位组织（bit organized）使得每一位都可以被重写。闪存是按块组织的，所以在更改信息前需要擦除整个块（扇区）。双向存储器是静态的，不需要刷新，数据的读出也是非破坏性的。通过控制向存储位元中写数据的电流脉冲的振幅，可以在多个非晶态（或非晶度）做出选择——每个状态都有不同的电阻。因此，双向存储器可以是多位存储位元（即每个位元可以存放多个位）的候选项，将促进位/芯片密度的提升。

双向存储单元能够承受10^{13}次改写而不发生退化，还可以通过不同程度的相变来存储多个值（类似MLC闪存技术）。双向存储器的主要优点是稳定性。在正常的操作温度下，据估计，存储单元的内容将保持稳定超过300年。同样，从读写次数的角度来说，相变设备的预期寿命比闪存单元的要长。

闪存、DRAM以电荷的形式存储数据很容易受电离辐射的影响，这妨碍它们在一些环境和军事上的应用。相变存储器设备具有固有的耐辐射特性。

双向存储器技术需要关注的一个问题是它的可扩展性（scalability）：设备可以被越做越小吗？由于存储位元需要加热装置来改变双向物质的状态，减小位元的大小会不会导致热量改变其邻居位元的状态？幸运的是，材料中热流量的理论计算表明，热量可以随着规模减小。即，当位元变得更小时，位元之间的热流量也会减少，因此可扩展性不是问题。

2009年6月，Samsung电子宣布与Numonyx（Numonyx已自2008年开始制造90nm 128Mb的PRAM，但规模较小）共同开发的相变存储器。2009年9月，Samsung宣布它已经开始制造针对移动电话市场的60nm512Mb的PRAM。2009年10月，Numonyx展示了在硅片上垂直堆叠多个相变存储器阵列的能力，并打算使用45nm技术生产容量为Gb量级的PRAM。来自Numonyx的表2-5说明了2010年相变存储器与其他存储技术相关参数的比较结果[⊖]。

表2-5 PCM与其他存储机制的比较

高密度存储器技术的比较

特性	DRAM	PCM	NAND	MLC NAND	HDD
非易失性	否	是	是	是	是
擦除操作	位	位	块	块	扇区
软件	简单	简单	复杂	非常复杂	简单
功耗	约1W/GB	100～500 mW/晶片	约100mW/晶片	约100mW/晶片	约10W
写带宽	约1GB/s	1～100MB/s/晶片	10～100MB/s/晶片	约10MB/s/晶片	200～400MB/s
写延迟	约20～50ns	约1μs	约100μs	约800μs	约10ms
写功耗	约0.1nJ/b	小于1nJ/b	0.1～1nJ/b	小于1nJ/b	大于10nJ/b
读延迟	50ns	50～100ns	10～25μs	25～50μs	约10ms
读功耗	约0.1nJ/b	远小于1nJ/b	远小于1nJ/b	远小于1nJ/b	大于10nJ/b
空闲能耗	约1W/GB	远小于0.1W	远小于0.1W	远小于0.1W	小于10W
读写次数	无穷	10^8	10^4～10^5	10^4左右	无穷
数据滞留	ms	永久	若干个周期	若干个周期	永久

⊖ S. Eilert, M. Leinwander and B. Crisenza, "Phase Change Memory (PCM): A new memory technology to enable new memory usage models." Micron Technology Inc, June 2011.

忆阻器

最奇怪的潜在存储设备是忆阻器（memristor，又叫记忆电阻），它在1971年首次提出。Leon Chua研究了电路理论中的3个经典无源元件：电阻、电感和电容，认为应该存在第四种元件：记忆电阻。当时，这一假说纯属猜想。

Chua表示，记忆电阻会表现出磁通和电量之间的关系，就像电阻表现出电压和电流之间关系一样。他认为，忆阻器会记得通过它的电流值，即使在电流消失以后。这样忆阻器就具有存储器可以记忆两个不同状态的属性，从而成为数据存储的候选。

这一理论猜想诞生30多年后，HP的Stanley Williams在2008年宣布发现了忆阻器原型。二氧化钛（TiO_2）半导体在纯态（类似硅）具有高电阻。如硅那样，加入杂质原子（称为掺杂原子）后会明显改变其电性能。然而，与硅不一样的是，掺杂原子在电场中不稳定；它们会根据电流的方向漂移。假设有一个薄的二氧化钛薄膜，只在一侧掺杂（即它类似一个由掺杂和未掺杂的TiO_2构成的三明治）。给薄的二氧化钛薄膜施加电场，使掺杂层的掺杂原子向纯TiO_2层迁移，从而降低其电阻。去除电场后，未掺杂层将会记住电场的持续时间和强度[⊖]。

第一个实验性的忆阻器是一个三端设备。忆阻器技术在数字世界中作为存储元件且在模拟世界作为神经网络的一个组成部分而具有发展潜力。忆阻器是否可以替代FRAM、MRAM或者双向存储器仍有待观察。然而，近年来，出现了一些其他形式的忆阻器。例如，出现了类似钛氧化物特性的材料（如某些聚合物薄膜）。同样，人们对除离子迁移之外的其他现象也进行了研究，如电子自旋。人们迫切需要新的非易失性存储器技术，这是因为闪存技术似乎即将达到其密度极限。

本章小结

存储器是计算机世界的“灰姑娘”。许多计算机用户都知道Intel处理器和AMD处理器之间的差异，但他们可能不会鉴别例如Rambus DRAM、DDR3 DRAM和SDRAM之间的差异。同样，他们可能没有意识到存储器系统设计的趋势和新的非易失性存储器日益重要的作用。

然而，没有存储器，世界上所有的快速CPU都是无用的，除非作为设备来计算出π值小数点后的数千位。今天，多媒体应用所需的数据无休止地增长导致了对巨大存储容量的需求。此外，手机和移动应用需要低功耗的存储器，MP3播放器和数码相机需要非易失性存储器。

本章介绍了构建立即访问存储器部件的相关技术和组织。与处理器技术相比，半导体存储器更容易受到速度、价格和性能之间权衡的影响。非易失性存储器可以在关闭计算机时保存数据，但需要在将数据写入存储器时花费很长的时间。DRAM的成本低、密度高，但其访问时间比SRAM的要长，而且需要更复杂的逻辑控制。SRAM速度快、易于使用，但它比DRAM更昂贵、密度更低。

本章还介绍了3种新的存储技术：FRAM、MRAM和双向存储器。这表明工程师们在不断地寻求那些能保持两种电状态从而可以用来存储数据的材料。

⊖ Strukov, Snider, Steward, Williams, “The missing memristor found,” *Nature,* Vol. 453,1 May 2008, pp. 80-83.

习题

2.1 下列用语（当应用于存储系统中）的含义是什么？

a. 随机访问；　b. 串行访问；　c. SRAM；　d. DRAM；

e. 非易失性存储器　f. 访问时间；　g. 大部时间为读存储器；　h. 周期时间

2.2 计算机具有 1MB 的存储空间。

a. 假设该空间按照字节编址，它需要有多少根地址线来寻址？

b. 如果该计算机有 16 位的数据总线，可以访问字节和 16 位的字，试给出用来选择字节 / 字的方式。

c. 该计算机存储器容量为 512KB，每个字 32 位，如果使用 64Kb 的 SRAM 芯片、每片 4 位宽，需要多少 RAM 芯片来实现该存储器？

2.3 某存储器部件的地址范围从 0x00400000 到 0x007FFFFF。其容量是多少？

2.4 *存储层次*（memory hierarchy）的含义是什么，为什么它对 PC 和类似工作站的设计者来说是个非常重要的概念？

2.5 本章开始时指出，存储器的易失性和只读性是一组矛盾。然而，在某个应用场合下，易失性只读存储器会非常有用。你能想到该种应用场合吗？

2.6 设计师可以使用两种不同的静态存储设备来构造具有 16 位 CPU 的计算机。两种存储芯片都可以容纳 2^{22} 位。如果需要构建 16 位数据总线和 128MB 存储容量的计算机，存储芯片要么是 256K 个 16 位的字组成，要么是 4M 个 1 位的字组成。哪种方案可以实现更为紧凑的存储系统？提示：考察单个芯片的引脚数。

2.7 某公司在 2012 年设计了一台计算机。假定在其发布时，存储器速度是 CPU 速度的两倍。据估算，CPU 速度每年提高 20%。同样，据估算，存储器速度每年提高 10%。问多少年后 CPU 将必须等待存储器提供数据？

2.8 为什么所有的 ROM 都是 RAM，但不是所有的 RAM 都是 ROM？为什么 SRAM 比 DRAM 更适合于构造 Cache？

2.9 试述当前最先进最高性能的个人计算机的主存储器、Cache 和硬盘的典型容量是多少？

2.10 一个静态存储位元至少需要 4 个晶体管、一个 DRAM 位元可以使用一个晶体管来保存一位数据。存储器可以以少于每位一个晶体管的方式来保存数据吗？你能给出一种一个晶体管存储多位数据的机制吗？

2.11 来自某商业 SRAM 数据表的时序图如图 P2.11 所示，口头解释其过程。

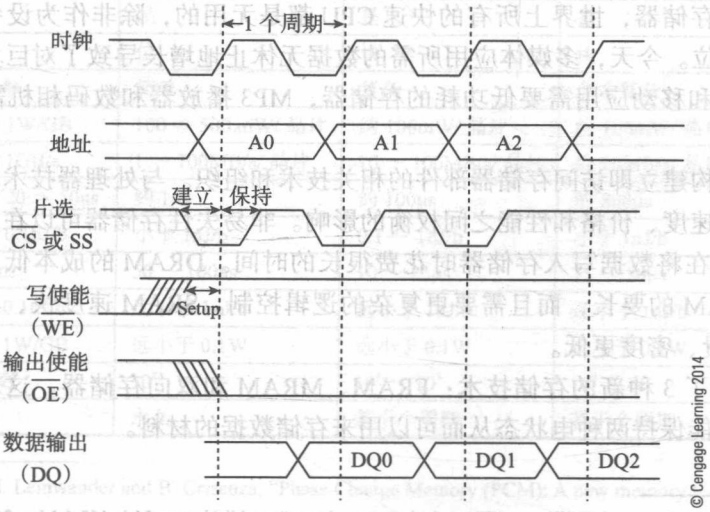


图 P2.11　SRAM 的时序图

2.12 SRAM 和 DRAM 的基本区别是什么？为什么 DRAM 要比相同大小（即按位计算的容量）的 SRAM 的引脚（I/O 连接数）要少？

2.13 传统 DRAM 中，列地址和行地址选通信号的作用是什么？在 DRAM 时序中，伪最大值（pseudomaximim）的含义是什么？哪个 DRAM 参数是伪最大值？对于设计工程师来说伪最大值的含义是什么？

2.14 双向存储器和铁电存储器之间的差异是什么？

2.15 实现存储系统所需材料的基本特性是什么？你可以想到一些尚未利用、但可以用来构建未来存储系统的材料特性并给出构建方式吗？

2.16 PC 中某 DDR SDRAM 模块的时钟频率为 133MHz，并连接到 64 位的数据总线。用存储系统的 PCxx00 标准来命名，其名称是什么？

2.17 DRAM 的速度每年增长约 7%，而处理器的性能每年提高约 60%。短期内这种状况的后果是什么？长期的后果又怎样？

2.18 某微处理器芯片有 16 位的数据总线，即它只执行 16 位的读和写操作。使用该处理器和两个 8 位存储器部件可以构造一个最小计算机系统。不幸的是，现在只有一个 8 位的存储器部件，这意味着不能直接建立一个完整系统。你有办法吗？请说明你如何使用一个 16 位的处理器和一个 8 位的存储部件构建一个完整的系统。

2.19 某计算机具有 24 位的地址总线，使用具有 $F=A_{23}\cdot A_{22}\cdot \overline{A}_{21}\cdot A_{20}\cdot \overline{A}_{19}$ 功能的逻辑电路来选择某个存储部件。通过该电路译码能够访问的存储地址范围是多少？

2.20 某计算机具有 64 位的数据字和 32 位的地址。通过 4 个体进行交叉。存储总容量为 1024MB。给出处理器的 32 位地址是如何划分的（即如何划分为 4 个域）？

2.21 某 DDR3 DRAM 被表示为 9-9-9-24。这是什么意思？

2.22 某计算机具有 64 位的数据总线，使用以下存储芯片。每种情况下，芯片表示为位置数 × 数据宽度。对于每一种芯片，说明所需的最小芯片数以及相应的存储容量。

a. 4M × 1；　　b. 1 M × 4；　　c. 256K × 16

2.23 如果可以在 DRAM 芯片中包含 Cache，你将如何组织它，可以获得什么优势？对计算机系统结构可能会带来什么样的变化？

2.24 某嵌入式微控制器具有 20 位的地址总线，可以访问以下 4 个内存块。试画出地址译码表以满足以下存储映射，并设计地址译码器来选中这些设备。

a. RAM1　0 0000—3 FFFF

b. RAM2　4 0000—7 FFFF

c. ROM1　E 0000—E 7FFF

d. ROM2　F 0000—F FFFF

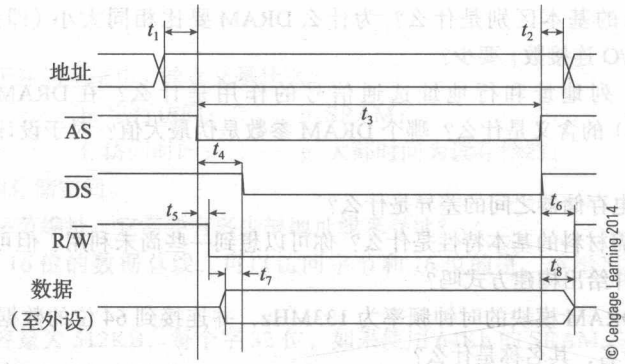
2.25 什么是损耗均衡，为什么需要这样做？

2.26 某 CPU 具有 24 位的地址总线和 16 位的数据总线，可以访问以下内存块：使用 256K × 8 位芯片构成的 1MB ROM 以及使用 2M × 4 位芯片构成的 8MB DRAM。试设计一个地址译码器来实现这样的安排。

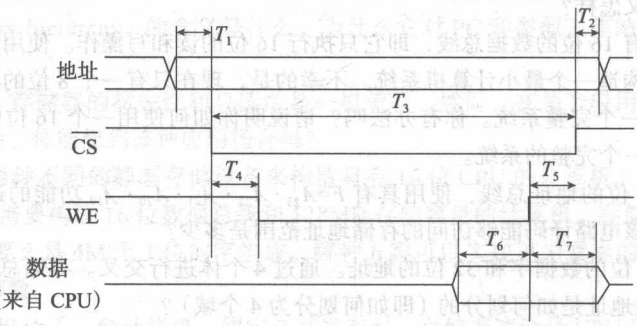
2.27 据报道，闪存的密度将达到极限。如果是这样，你有什么考虑？

2.28 如果需要为深空飞行器设计计算机。需要考虑哪些特殊因素，这对计算机的设计会产生什么影响？

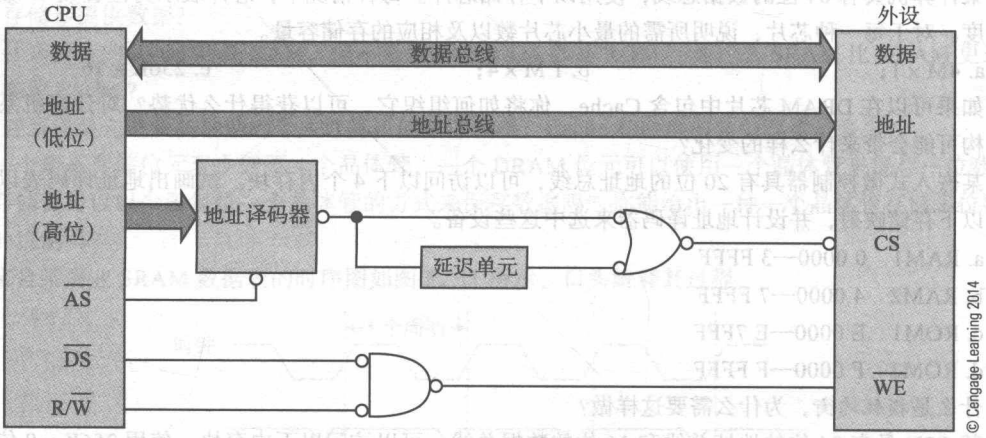
2.29 图 P2.29a 给出了微处理器写周期的时序图。图 P2.29b 给出了存储器映射的外设写周期的时序图。利用表 P2.29 中的时间信息，验证图 P2.29 接口功能的正确性。注意，存储器映射的外设在处理器看来就像 SRAM 一样访问（第 4 章将介绍这种外设）。本题中，低电平有效的信号用星号而不是上划线表示，例如，W* 表示非写。



a）处理器时序图



b）外设时序图



c）CPU 与外设接口电路

图 P2.29

注意：

$\overline{AS}$为地址选通信号（当来自 CPU 的地址有效时变为低电平）

$\overline{DS}$为数据选通信号（当来自 CPU 的数据有效时变为低电平）

R/$\overline{W}$为读 / 非写信号（读周期为高电平，写周期为低电平）

$\overline{CS}$为片选信号（当外设处于读或写周期时为低电平）

$\overline{WE}$为写使能信号（当外设处于写周期时为低电平）

表 P2.29

参数	名称	最小值 /ns	最大值 /ns
a）处理器时序			
t_1	地址建立	5	
t_2	地址保持	8	
t_3	地址选通有效	50	
t_4	地址选通为低电平至数据选通为低电平	10	
t_5	地址选通为低电平至 R/$\overline{W}$* 为低电平	10	15
t_6	地址选通为高电平至 R/$\overline{W}$* 为高电平	0	5
t_7	数据建立时间	2	4
t_8	数据保持时间		3
b）外设时序			
T_1	地址建立	2	
T_2	地址保持	1	
T_3	片选有效	20	
T_4	$\overline{CS}$* 为低电平至$\overline{WE}$*（写使能）为低电平	5	
T_5	$\overline{WE}$* 为高电平至$\overline{CS}$* 为高电平	2	
T_6	数据建立时间至$\overline{WE}$* 为低电平	10	
T_7	数据保持时间至$\overline{WE}$* 为高电平	2	

二级存储器

“整体的效果大于各部分的总和。”

——Anon

“我们像建设城市一样制造计算机系统：耗时、没有规划和继承性。”

——Ellen Ullman

“我现在就能看到未来，那时将没有可移动部件。”

——Bartoz Kijanka

“磁带将比我们所有人活得长。”

——某 HP 白皮书的题目

“不要把 CD 放进洗衣机——但是如果要坚持这样做，请使用为精致面料设置的机洗程序。”

——Steve Meretzky

处理器技术的快速发展使得计算机具有实时处理视频数据的能力，半导体技术的进步为现代 CPU 处理数据提供了大容量的直接访问存储器。本章将介绍存储层次中的下一个层次——二级存储单元（secondary storage unit），它用来保存当前没有被处理的大量数据。这些存储设备包括硬盘、固态硬盘以及 CD/DVD/ 蓝光光盘等。

当硬盘在 PC 中首次出现时，其容量仅为 5MB。今天，处理器就可能具有 8MB 的 Cache，3 个 TB[⊖]的硬盘已经成为商业化产品，2011 年 Samsung 成为首家展示容量为 4TB 硬盘的公司。Toshiba 公司宣布，计划在 2013 年以面密度 1Tb/in^2 实现 5TB 的硬盘。20 年间，硬盘的容量已增长了 80 万倍。另一方面，硬盘访问时间的变化没有跟上其容量增长的速度。今天的硬盘访问时间仅比 10 年前稍快一点，而对固态硬盘的研发使新一代高速磁盘的出现成为可能。

由于二级存储器对计算机的发展至关重要，本书使用一章来介绍其技术和特点。首先讨论磁记录过程的性质并说明光盘驱动器操作过程。特别是将展示其机电特性如何决定其性能。通过引入固态盘（solid state disk，SSD），当前的半导体技术已经渗透大容量存储器市场，故本章将对其进行介绍。

本章的第二部分将讨论光学存储介质的设备，包括 CD、DVD 以及蓝光光盘，它们提供了低成本、非易失性、可移动的二级存储能力。这些介质的工作非常类似磁盘，只是其使用的是物质光学特性的变化而不是磁特性的变化来记录数据。

⊖ 1MB=2^{20}B，1GB=2^{30}B，1TB=2^{40}B。如果某小说有 10 万个单词，假定平均每个英文单词由 5 个字母组成，那么 3TB 的硬盘可以放下 600 万（$3\times2^{40}/2^{19}$）部小说。

3.1 磁盘驱动器

1956 年，IBM 发明了第一个硬盘驱动器并作为 305 RAMAC 系统的一部分。它的盘片直径为 24 英寸，容量为 5MB，体积比一台洗衣机还大，成本为几千美元。到了 1983 年，Seagate 发明了第一款 PC 用磁盘驱动器，它能够存储 5MB 的数据而成本仅为 1500 美元，并可以放进 PC 的内部。今天，硬盘的容量已达 4TB，成本不足 50 美元。

磁盘驱动器使用的技术人们早在 20 世纪 40 年代就已经了解。事实上，磁盘是 1877 年爱迪生所发明留声机的直接衍生物。爱迪生将声音保存在圆筒上由锡箔（后来为蜡）覆盖的轨道上。爱迪生留声机根据声音震动在物理上改变凹槽的形状从而实现声音的存储，而磁盘通过对轨道表面进行磁化来存储数据。CD/DVD/ 蓝光光盘通过改变轨道表面的光学性质来存储数据。

磁盘驱动器具有旋转的盘片，表面覆盖着一层非常薄的材料，它可以被磁化为两个方向之一：北指向南或者南指向北。旋转盘片位于写磁头下方，磁头使盘片表面被磁化形成一条由 1 和 0 组成的圆形轨道。当需要检索数据时，读磁头（通常与写磁头共存）检测表面的磁通，并据此重建所记录的数据。原理似乎非常简单。

实际上，由于磁记录区域非常小且磁盘高速旋转，真正磁盘驱动器的结构和操作是非常复杂的。现代磁盘驱动器的细节真是可怕：磁层的厚度仅为 2000 个原子，读写磁头在距离磁盘表面 0.2μm 处高度旋转。在磁层的顶部是一层碳氟化合物组成的润滑层，只有约一个分子的厚度。

在介绍磁盘驱动器的结构和特点之前，本章首先讨论磁记录的原理。包括近年来促使磁盘容量从 10GB 发展到 4TB 的磁盘驱动器设计技术，这是一个里程碑。来自 IBM 的图 3-1 给出了几种磁盘驱动器的面密度（areal density）及对应年份，并绘出了从 1956—2001 年这 45 年中磁盘记录密度显著增加的情况。面密度指出了磁盘的位记录密度，通常以每平方英寸记录的位数来衡量。面密度从约 2×10^{-3}Mb/in^2 发展到 4×10^{4}Mb/in^2，在约 50 年的时间内增加了 7 个数量级。到 2010 年，Toshiba 发布了面密度为 540Gb/in^2（即 54×10^{4}Mb/in^2）的硬盘，一年后 Toshiba 发布了一款在笔记本中使用的硬盘，其面密度为 744Gb/in^2。

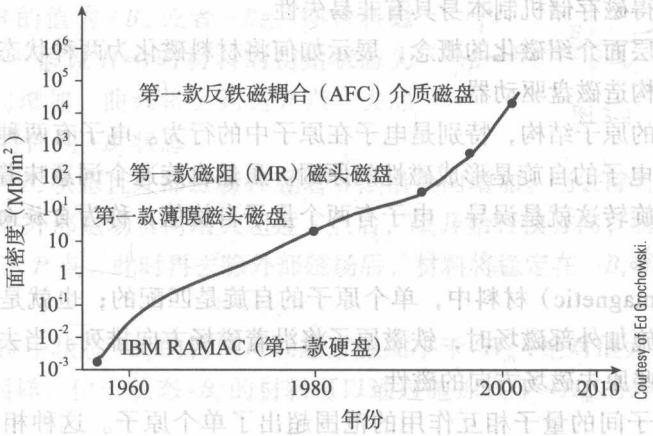


图 3-1 1960 年以来的记录密度

磁盘容量随着微软操作系统的发展而不断提升。图 3-2 给出了磁盘容量和操作系统的发展情况。没有大容量磁盘，不可能出现今天的操作系统。当然，也有些人认为，今天的许多

软件都是臃肿的，应该使用较小的磁盘空间。

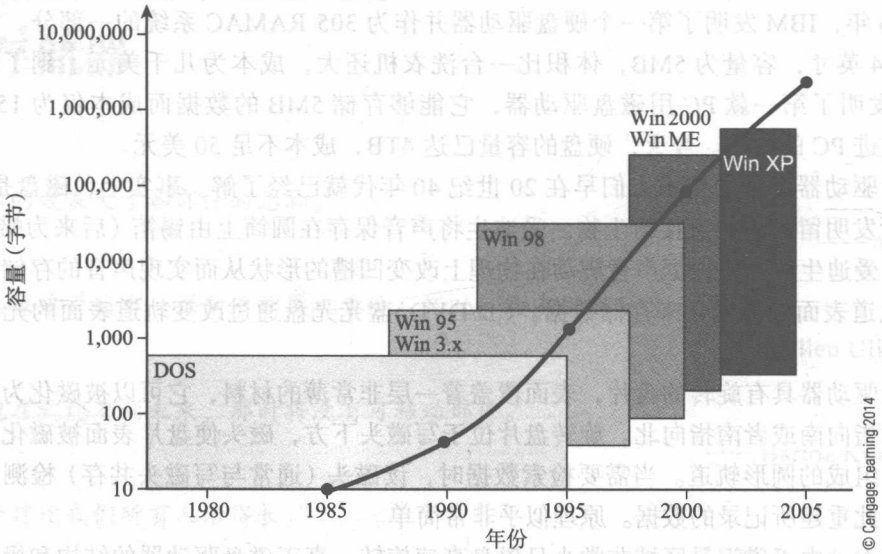


图 3-2 操作系统、磁盘容量随时间的关系

3.2 磁性和数据存储

磁存储技术相当有趣，它属于计算机技术的早期技术，依赖具有运动部件的机电装置一起工作。磁记录技术已经被使用了很长一段时间。例如，钢丝录音机可以在一卷钢丝上录制声音，直到二战后才被磁带录音机取代。然而尽管存在固有的局限性，磁存储技术始终没有消失。事实上，2000 年 12 月《IEEE Spectrum》中的一篇文章的题目就是“磁存储器：不会消亡的介质”。[⊖]

材料的磁性可能是最明显的数据存储方式，因为磁性是很好的二进制记录介质：磁性粒子可以磁化为 N-S 或者 S-N 方向。当某材料被磁化后，它将保持磁化状态直到按照相反的方向被磁化，这使得磁存储机制本身具有非易失性。

本节将在原子层面介绍磁化的概念，展示如何将材料磁化为两种状态。然后说明如何使用材料的这种属性构造磁盘驱动器。

磁性源于材料的原子结构，特别是电子在原子中的行为。电子有两种动作：绕原子核旋转与自旋（spin）。电子的自旋是形成磁性的原因。虽然自旋这个词意味着旋转，如果读者认为电子像陀螺一样旋转这就是误导。电子有两个量子自旋值，称为自旋向上（spin up）和自旋向下（spin down）。

在铁磁（ferromagnetic）材料中，单个原子的自旋是匹配的；也就是说，相邻原子之间是相互作用的。当施加外部磁场时，铁磁原子将沿着磁场方向排列。当去除外部磁场后，铁电材料可以保留一些原先磁场方向的磁性。

铁磁材料中电子间的量子相互作用的范围超出了单个原子。这种相互作用导致称为域（domain）的某个范围内的原子磁矩并行排列。域的大小为 30nm ～ 150μm。

铁磁材料中，域的排列是随机的，如图 3-3a 所示，整体上没有磁性。图 3-3b 给出了施

⊖ R. Comerford. “Magnetic Storage: The medium that wouldn’t die.” IEEE Specturm, December 2000, pp. 36-39.

加外部磁场后的影响。与外部磁场方向相同的域保持其磁化方向，而与外部磁场方向不同的域将根据外部磁场旋转而改变其方向。

由于材料内部磁场是外部磁场和所有域对应磁场的叠加，当越来越多的域改变为与外部磁场相同的方向，内部磁场的磁性将迅速增加。突然，内部磁场像雪崩一样建立起来，与外部磁场方向相同的域急剧增加。很快，所有域都磁化为相同的方向，如图 3-3c 所示，此时材料称为被磁化（magnetize）。如果去除外部磁场，材料仍保持磁化状态，这是因为由域组成的磁场足够强大可阻止域的重新排列。

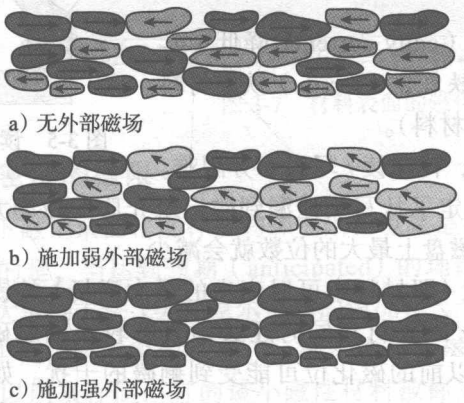


图 3-3　磁化和域

提高材料的温度增加了原子的热运动。在磁材料中，不断加剧的热运动最终将克服域的凝聚力从而使材料消磁，域的方向再次变为随机。发生此种变化的温度称为居里点（Curie point），铁的居里点超过 1000℃。

铁磁材料的内部磁场相对外部磁场的变化可绘出如图 3-4 所示的曲线，该曲线被称为磁滞曲线（hysteresis curve）。

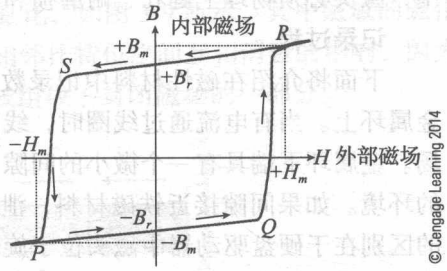


图 3-4　磁滞曲线

横轴 H 代表了外部磁场。没有外部磁场时（即 H=0），内部磁场 B 的值为 $+B_m$ 或者 $-B_m$。即材料磁化为两个状态之一。假设 H=0 时材料的初始状态为 B=$+B_r$，H 沿着正向增加。曲线将会到达 R 点。去除外部磁场后，材料将回到 $+B_r$ 状态。

假设材料处于 $+B_r$ 状态且外部磁场 H 朝着 S 点沿负向增加。当去除外部磁场后，材料将回到 $+B_r$ 状态。如果外部磁场负向增大超过 S 点后，域开始转换方向，材料将迅速改变其磁化方向，曲线将到达 P 点。此时再去除外部磁场后，材料将稳定在 $-B_r$ 状态，只是方向与原来相反。

图 3-4 表明，位于状态 $+B_r$ 的材料可以通过施加小于 $-H_m$（绝对值大于 H_m）的外部磁场而变为状态 $-B_r$。同样，位于状态 $-B_r$ 的材料可以通过施加大于 $+H_m$ 的外部磁场而变为状态 $+B_r$。

可以通过施加足够强的正向或负向磁场使材料磁化。下面将解释如何在实践中实现这种操作以及如何检测磁性材料的状态。

磁性材料的另一个特性是它的矫顽力（coercivity），即将残余磁通量减小到零所必须施

加的磁场强度。磁性材料通常分可以为两类：硬（hard）磁材料和软（soft）磁材料。硬磁材料具有高矫顽力，可以被永久磁化；而软磁材料具有低矫顽力，当去除外部磁场时不表现出磁性。磁盘中的读写头是由软磁材料构成，而存储数据的磁性涂层由硬磁材料构成。

3.2.1 读/写头

图3-5给出了在磁记录介质（如硬盘、软盘和磁带）上读写数据所使用的读/写头的结构。早期的记录磁头由环形软磁铁磁体绕着几圈线圈构成。

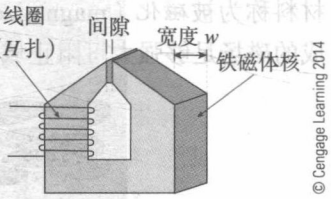


图3-5 读写磁头的基本结构

高频磁场在写头中产生的涡流（eddy current）降低了其工作效率。第二代头使用不导电的铁氧体（ferrite）铁磁材料（由铁和镍或铁和锌组成的陶瓷复合材料）。

由于记录系统复杂的访问需求，构建读/写头十分困难。磁头间隙必须尽可能小，因为它决定了从写头泄漏到磁记录介质的磁场大小。如果间隙很大，磁化的面积也大，可以存储在磁盘上最大的位数就会减少。

磁头必须使用具有极高饱和度（即材料中可以产生的最大磁场）的铁磁材料构成，这是因为磁化记录介质需要强磁场。磁头材料同时还必须具有较小的剩磁（即写电流消失后的剩余磁场）。如果剩磁太高，以前的磁化位可能受到剩磁的干扰。如果使用同一个磁头来完成读写，它必须具有高渗透率（permeability），人们利用材料的渗透率来衡量材料对磁通的导通能力：渗透率越高，就越容易磁化材料。读头也应该具有低饱和磁致伸缩（magnetostriction）。磁致伸缩用来描述磁场变化改变材料尺寸的现象。如果读头遭受任何形式的物理冲击，磁致伸缩将产生寄生磁场，导致线圈中出现杂散电流。所有这些磁性特性中，磁头必须物理上健壮、耐磨损和耐腐蚀。

记录过程

下面将介绍在磁性材料中记录数据的过程。图3-6给出了记录过程的示意图。线圈绕在金属环上。当有电流通过线圈时，线圈中就会产生磁场，这进一步导致了在金属环中产生磁场。金属环下端具有一个微小的间隙，磁场会通过该间隙。实际上，泄露的磁场改变了周围的环境。如果间隙接近铁磁材料，泄漏的磁场将使其磁化。硬盘与磁带（或卡带）记录装置的区别在于硬盘驱动器中磁头位于旋转的盘片上方，而磁带驱动器中磁头位于涂有磁性材料的磁带上方。

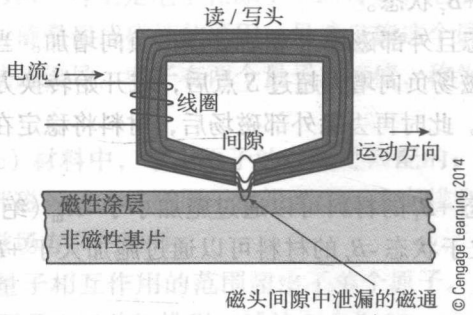


图3-6 记录过程

图3-7展示了改变写磁头线圈中的电流所产生的影响，以及通过写头下方的材料表面的

磁化情况。

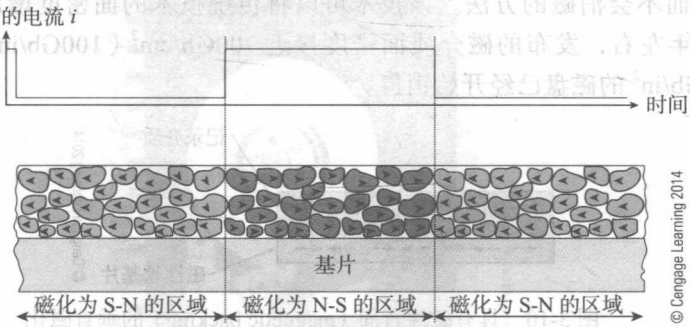


图 3-7 材料表面的磁化

3.2.2 磁记录密度的极限

磁记录机制下磁记录面密度存在物理极限；也就是说，每平方英寸可存储的最大比特数是有限的。幸运的是，当达到预期（anticipated）的理论限制后，物理学家和工程师似乎又找到扩展理论最大值的方法（类似摩尔定律的情况）。

20 世纪 90 年代，科学家们认为，一种称为*超顺磁*（superparamagnetic）的效应限制了磁记录密度。可以用来存储信息的最小磁性材料被称为*磁颗粒*（grain，一种单磁畴粒子），通过其构成了如磁盘表面的磁性材料。当颗粒达到 10nm 左右的量级时，热效应会导致这些颗粒在室温下自动消磁。这导致最大面密度约为 6Gb/cm^2 或 0.93Gb/in^2[⊖]。幸运的是，人们已经发现了几种避免超顺磁效应的方法。

图 3-8 展示了在记录介质表面水平磁化的效果。该图表示以最小区域进行记录的最坏情况，交替记录了 N-S 和 S-N。另一种记录方式是垂直磁化，如图 3-9 所示，其中磁域的磁化方向与记录介质表面形成直角。垂直记录方式减少了相邻比特位之间互相消磁的影响，因为某域的朝向并没有与另一个域的朝向针锋相对。这些域组成了封闭磁场的一部分。

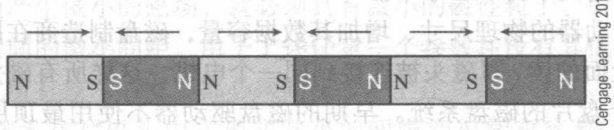


图 3-8 传统的水平磁化

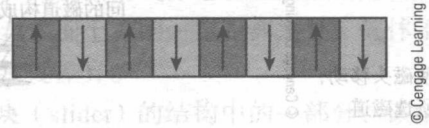


图 3-9 垂直磁化

垂直记录需要特殊的写头设计（即与传统的水平磁化记录磁头不同）。需要使用“单极型磁头”在介质上进行垂直磁转换写操作。

⊖ 1in=0.0254m。——编辑注

磁微粒的微小距离会使相邻粒子彼此消磁。图 3-10 显示了一种由 Fujitsu 提出的减少垂直粒子的大小而不会消磁的方法。该技术可以将传统技术的面密度增加 8 倍，达到 50Gb/cm^2。到 2000 年左右，发布的磁介质面密度接近 700Gb/cm^2（100Gb/in^2）。到 2011 年，面密度超过 700Gb/in^2 的磁盘已经开始销售。

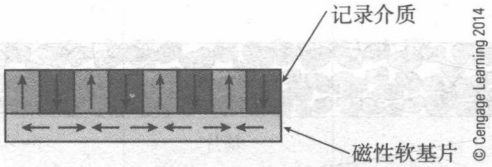


图 3-10 具有磁性背面（magnetic backing）的垂直磁化

延缓不可避免的超顺磁效应产生影响的一种方法是，通过增加反转磁颗粒状态所需的能量屏障来改进介质的磁特性。图 3-10 中的方法是在磁记录表面下设置一层磁性软介质。

传统的磁面是均一的；也就是说，它们是由单一金属薄膜或由磁性颗粒黏和在一起的涂层构成。硬盘设计师不断探讨使用晶格介质（patterned media）来存储数据。每一位晶格由被非磁性介质包围着的磁性材料组成，这样可以降低相邻区域间的消磁效应。

3.2.3 磁盘数据记录原理

下面将介绍数据是如何存储在磁盘中的，以及记录过程和回放过程是如何工作的。图 3-11 显示了数据在磁盘表面上是如何组织的。读 / 写头可以向中心或者外围移动或步进（step）。当磁盘旋转时，磁头下的轨迹形成磁道（track）。对于单位数据来说磁道容量太大，故磁道通常被分为不同的扇区（sector）。一个扇区是从磁盘读写数据的最小单位。磁盘上数据的组织结构对磁盘驱动器的性能有重要的影响。以数据的粒度（granularity）为例来说明：如果使用小扇区，由于大文件会占用许多扇区，每个扇区都有开销，因此这种方案的效率不高；另一方面，如果使用大扇区，当用户需要存储小规模数据时，该方案的效率也不高。例如，如果扇区大小为 8KB，用户具有许多 3KB 大小的文件，每个扇区就会浪费 5KB 的空间。典型的磁盘扇区的大小为 512 字节。

为了减少磁盘驱动器的物理尺寸、增加其数据容量，磁盘制造商在同一个转轴上安装几个盘片，读取每个盘面的读 / 写磁头被连接到同一个电机，这样所有磁头可以一起移动。图 3-12 给出了具有 3 个盘片的磁盘系统。早期的磁盘驱动器不使用最顶层和最底层的盘面存储数据，因此图 3-12 中只使用了 4 个盘面。然而，现代磁盘驱动器使用所有盘面。

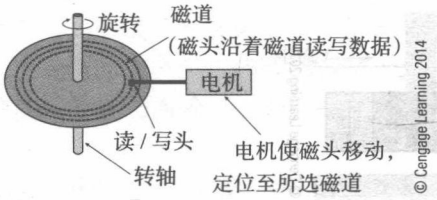


图 3-11 对材料表面的磁化

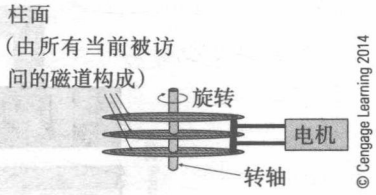


图 3-12 多个盘片的磁盘驱动器

图 3-12 中，电机使磁头在磁道之间移动，磁头组件沿着磁道移动。这种方式如今没有广泛使用是由于其实现复杂、速度慢。一种更简单的电机使用磁臂在磁盘表面上扫动（就像古老的黑色乙烯留声机的唱臂）。图 3-13 给出了一款使用类似磁头追踪机制的 Seagate 硬盘驱动器的照片。

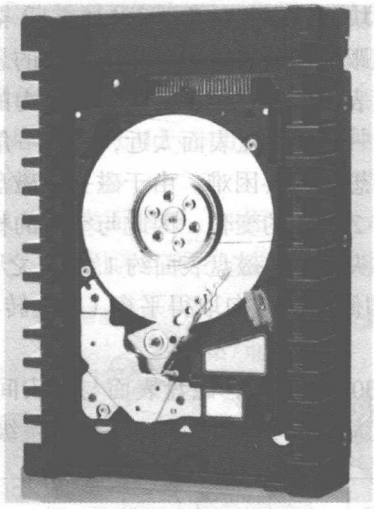


图 3-13 磁盘驱动器

盘片尺寸

微机中标准的硬盘为 3.5 英寸驱动器。读者可能认为其盘片的尺寸小于 3.5 英寸。其实不然。其盘片的直径为 3.75 英寸。同样，3.5 英寸驱动器的宽度并不是 3.5 英寸，而是 4 英寸。3.5 英寸这个术语表示的是磁盘的形状因子（form factor）而不是其物理宽度。

磁盘速度

磁盘的边缘旋转到底有多快？

若磁盘的直径为 3.75 英寸，则其周长为 11.78 英寸。如果磁盘转速为 7200rpm（每分钟的转数），磁盘外沿的速度为 7069 英尺 /min 或 424 千英尺 /h。也就是 80.3mph（每小时英里数）。

为了在磁盘表面产生微小的磁场，就必须具有微小的磁性粒子、微小的磁头间隙以及在磁头和记录介质表面间的微小间距。由于上述任意一个参数在没有其他参数配合的条件下都不能得到提升，因此所有技术的发展都必须同步考虑上述参数的改进。

使用现代制造技术在写磁头中实现非常小的间隙变得相对容易。实现以 50mph 旋转的盘片并使磁头保持在盘片上方仅有 10×10^{-9}m 有点棘手。事实上，如果没有边界效应（boundary effect），这是不可能实现的。

当磁盘在空气中旋转时，其表面的气体也会以磁盘的速度旋转，这是由于粗糙的磁盘表面拖动了气体分子。但在磁盘表面上方的一定距离之外，气体不会移动。因此，在磁盘表面和表面上部空间之间产生了速度差异。

读 / 写头构成了称为滑块（slider）的结构中的一部分，该结构主要完成将磁头移动至所需磁道的功能。滑块将磁头和电机通过悬挂臂（suspension arm）连接在一起，悬挂臂用于传输来自磁头的信号，并将磁头保持在所需位置。滑块具有合适的气动特性，它可以“飞行”在处于边界效应的磁盘表面上方。

机翼（这里为滑块）产生的升力与其表面空气的流动速度的平方成正比。当滑块即将移动到磁盘上时，空气流动得更快，从而产生更大的升力，使得滑块几乎保持在磁盘表面上方

一个恒定的位置。也可以说，滑块通过空气轴承（air bearing）悬停在磁盘表面。

飞行头能够以非常高的精度跟随磁盘表面的起伏。滑块的悬架必须提供一个向磁盘方向的力来对抗会导致滑块飞离磁盘表面的空气轴承的气动力。力的大小必须精确地施加在恰当的位置，否则扭转力会使滑块的一角离磁盘表面太近，而另一角离磁盘表面的太远。

设计有效的空气轴承比人们想象的要困难。由于磁头在磁盘的内道和外道之间移动，磁头与磁盘表面的相对速度大约有 2 ：1 的变化，因此与空气的相对速度也在变化。现代空气轴承设计能够弥补这一点，使磁头保持在磁盘表面约 1.0 μin 之上的距离。

悬架必须允许滑块在俯仰和侧滚的方向取得平衡（即旋转），这样尽管磁盘表面有起伏都可以使其与表面保持接近。

图 3-14 显示了从 1993 ～ 2004 年之间，磁盘表面与磁头间距的减少情况。这种改善表明当代磁头距离磁盘表面的高度大约为 10nm（即 10^{-8}m）数量级，这是一个非常小的间隙。

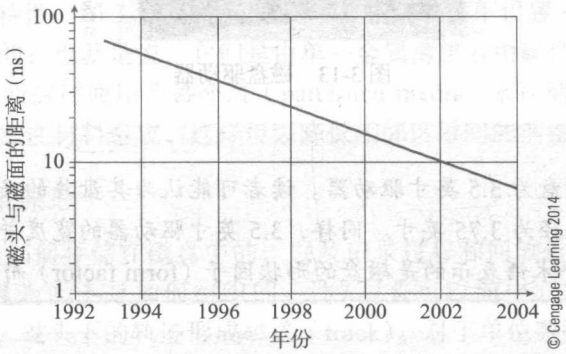


图 3-14 磁头与磁面的距离

如果磁头未能保持在磁盘表面上方，以约 50 英里 /h 的速度撞击磁盘，将导致磁性涂层的损害并破坏数据。该情况被称为*磁头碰撞*（head crash），这个词现在用于指示所有突发、灾难性的计算机系统故障。图 3-15 给出了经典的比较，它表明从读 / 写头到磁盘表面的间距是多么小。按照相同比例，可以对比该间距与人类头发、烟雾颗粒和指纹的大小。是的，磁头的飞行高度比指纹的高度还小。

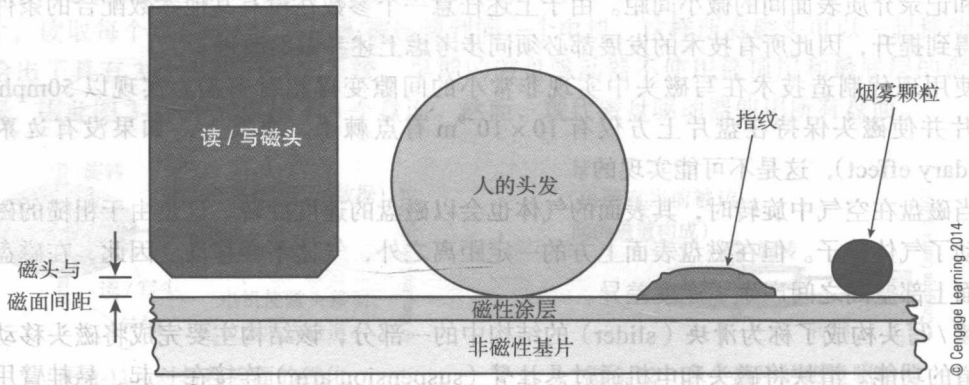


图 3-15 磁头与磁面间距相对大小

如果设计桌面计算机硬盘中具有微小的飞行高度和轨道间距的磁头悬架机制是一件困难工作的话，想想那些为笔记本电脑和其他便携电脑设计悬架、滑块以及磁头机制的工作。磁头必须在因计算机移动产生震动和加速运动的情况下保持其飞行高度。

磁盘驱动器不工作时磁头的状况又如何呢？早期的磁盘驱动器在启停阶段使用着陆区（landing zone）来放置磁头。当然，该区域并不用来存储数据。磁头和磁盘表面技术的改进导致了非常光滑的磁头和光滑的磁盘表面。非常非常光滑，以至于当磁盘停止旋转时，磁头和磁盘表面可以粘在一起（这是冷焊（cold welding）的一种形式，发生在两个原子级光滑表面接触时）。

今天，磁头制动并将其从磁盘表面移开，如图 3-16 所示。当关闭电机的电源，它在减速时产生反电动势（back EMF）。使用该电压可以将滑块移动至某个斜面并脱离磁盘的表面。

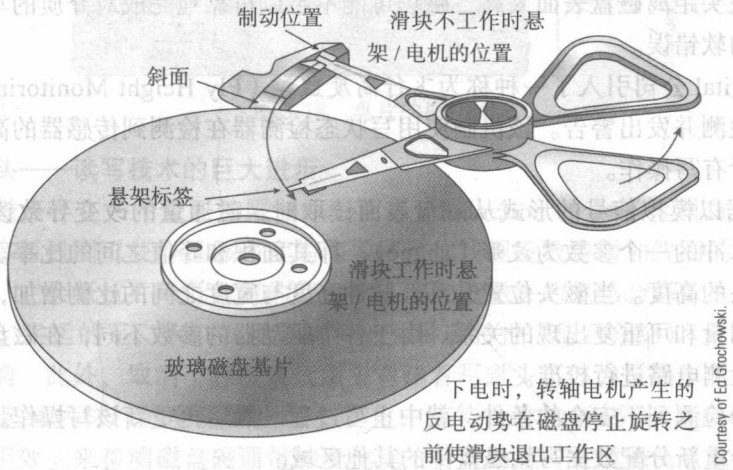


图 3-16 磁头工作机制

磁盘可以存储的数据量是每英寸磁道的数量以及磁道中每英寸记录的数据位的函数。这两个参数的乘积就是面密度（areal density），用每平方英寸记录的位数表示。只能通过提高磁道中每英寸记录的数据位或每英寸磁道的数量来提高面密度。当每英寸记录的位数增加时，每位所占的空间相应减小。如果每位变得更小，磁盘旋转导致磁头读出的信号降低，这使得对数据进行可靠的解码变得更困难。

热校正

一些早期的磁盘驱动器使用步进电机（step-actuator），它将磁头沿直线一次移动一个固定的步长。那时，磁盘的磁道间距离相对较远，跟踪磁道并不是主要问题。旋转电机（rotary actuator）通过旋转一个给定的角度使磁臂在磁盘上移动，它现在已经取代了步进电机。今天的磁道间距比几年前更加紧密，将主轴旋转一个精确的角度比移动磁头还要困难。磁头定位精度和磁道跟踪成为比过去更主要的限制因素。

情况还可能变得更糟。当磁盘驱动器的各个组成部分升温或降温时，它们的变化速度是不一样的，导致磁头不能精确定位磁道。某些磁盘驱动器使用了一种称为热校正（thermal calibration）的操作，周期性地对磁头进行重新调整和重新定位。可以在驱动器的温度变化超过预定值或者间隔一段时间后进行重新校正。此操作可能会导致在磁盘和主机控制器之间的数据流出现一个约 500ms 的间隔。这样一个间隔对数据流来说通常并不明显，但它对视听应用来说是不可接受的，这是因为声音或运动视频的短暂停顿都是令人烦恼的。20 世纪 90 年代中期，当视听技术逐渐流行时，磁盘制造商生成了一些 AV

硬盘，其中的热校正过程被有效地隐藏。

今天的磁盘引入了反馈来实现更精确的磁头定位机制，因此没有必要再进行热校正了。

使用读/写头来最大化面密度的一种方法是减少磁头到磁盘的距离——减少磁头的飞行高度（fly height）。降低飞行高度使得每位的输出信号更强、更容易被检测。

不幸的是，有几个因素会对飞行高度产生不利影响，例如纬度或温度的变化、污染物的影响、外部冲击和振动等。磁盘环境参数的优化可以在读取数据时降低驱动器的错误几率。同样，如果写磁头距离磁盘表面太高，磁场可能不足以可靠地完成对介质的写操作。这将导致可以被纠正的软错误。

Westen Digital公司引入了一种称为飞行高度监控（Fly Height Monitoring）的机制对潜在的失效进行检测并发出警告。该机制使用写状态检测器在检测到传感器的高度偏离其正常位置时就暂停所有写操作。

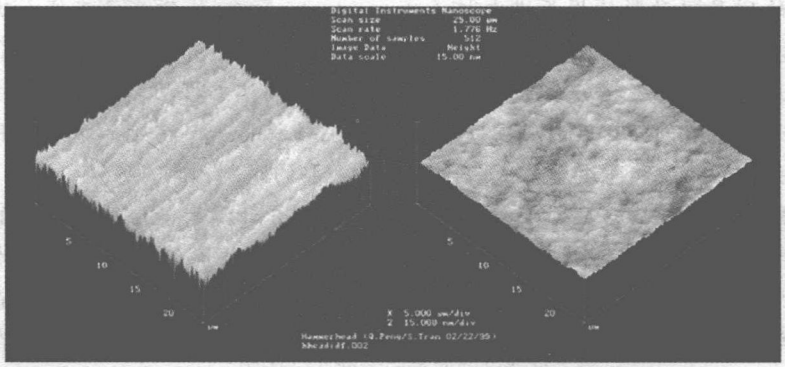
当原始数据以模拟信号的形式从磁盘表面读取时，磁通量的改变导致读磁头中产生脉冲。用来定义脉冲的一个参数为波形（shape）；即其面积和峰值之间的比率。脉冲的波形可以用来监测磁头的高度。当磁头位置升高，脉冲高度与宽度之间的比例增加，这为每个磁头提供了一种可测量和可重复出现的关系。由于各个驱动器的参数不同，在磁盘制造过程中需要对磁头高度检测电路进行校准。

如果监控器检测到不安全的条件，就中止写过程。然后再重新该写操作。如果写操作仍不成功，可能会重新分配数据写到磁盘中的其他区域。

1. 盘片技术

今天的盘片比其祖先更加复杂，早期的盘片大都是铝盘片表面黏和磁性材料（铁氧化物）构成的涂层。影响磁盘数据密度的两个关键参数是读/写磁头的飞行高度和磁性粒子的大小。

某些磁盘是由低杂质的玻璃制成，这是由于玻璃比铝更耐热（其膨胀系数更小）、更光滑、更坚硬。图3-17是来自IBM的显微照片，比较了铝和玻璃表面的光滑度。相同重量下，玻璃比铝更坚硬。较好的刚度将在高速运动时降低噪声和振动。玻璃的刚度使得盘片可以做得更薄更轻，这可以减少主轴电机的负载。此外，更轻的盘片意味着更短的加速时间。



Composed from two original images © IBM Corporation. Images used with permission.

图3-17 盘片表面的光滑度

磁盘表面被溅射（sputtering）了多个涂层。溅射过程是将磁盘放在高真空中，然后通过蒸发涂层在磁盘表面创建一层薄膜。

现代磁盘片包含 5 层或更多的层（如图 3-18 所示）。最上层为润滑层，提高磁头与磁盘接触的耐久性。润滑层下面是一层薄的碳基保护层。润滑层厚约 1nm，保护层厚约 15nm。记录表面由两层组成：记录层（通常是钴和铬的化合物）和铬底层。最后，玻璃基片为上述 4 层提供衬底。

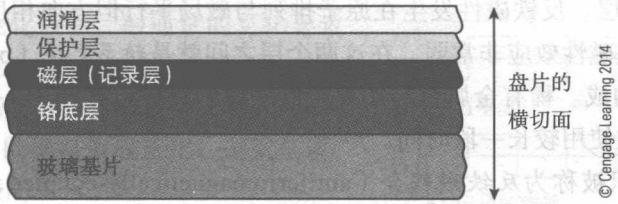


图 3-18　盘片的横切面

2. 巨阻磁头——读写技术的巨大进步

传统的读磁头存在一个重要的限制——电感（induction）。为了检测磁通的变化，磁头上需要缠绕许多匝线圈以获得足够强度的信号。增加线圈匝数将增加线圈的电感。电感是一种电路的属性，它会抵抗并降低电流的变化率。增加读磁头的电感会降低从磁盘表面读出的磁通的变化率。幸运的是，人们发现了材料的一种磁特性（稍后介绍），可以极大降低电感对读磁头的影响。此外，取消了感应式读磁头意味着写磁头可以为写操作进行优化。

磁场会引起某些材料的电阻发生微小变化，该特性称为*磁阻*（magnetoresistive，MR）效应。使用磁阻效应来检测磁盘磁通量的变化优于传统的感应式读磁头，这是由于磁阻磁头具有更小的电感，可以更快地读取数据。

IBM 在 1991 年的 1GB 磁盘驱动器中率先使用磁阻磁头，到 1994 年，IBM 的实践已经表明达到 3Gb/in^2 的面密度是可能的。不幸的是，磁阻磁头输出的电信号非常低。20 世纪 80 年代后期，研究人员发现某些材料在磁场中的电阻会发生将近 50% 的巨大改变。这种属性被称为*巨磁阻*（giant magnetoresistive，GMR）效应，它存在于由很薄的金属元素层交替组成的材料中。人们很快就意识到，巨磁阻效应可以用来建立高效的读磁头，IBM 是第一个尝试利用巨磁阻效应的商业组织。

巨磁阻磁头的优点是它们对磁盘的磁场变化更加敏感，可以检测尺寸更小的记录位。同样，数据可以以更高的速度读取且相比磁阻磁头来说电噪声被降低。大约在 1998 年，IBM 使用巨磁阻磁头使得面密度超过 11.6Gb/in^2。这些磁头使用感应器的厚度为 0.04 μm，IBM 声称如果将感应器减半至 0.02 μm 则可能达到 40Gb/in^2 的密度。高记录密度的优点是磁盘的物理尺寸和功耗可以减少，进而提高数据传输率。在给定容量下，如果磁盘较小且读 / 写头较轻，则主轴转速可以进一步增加，磁头机械运动所造成的延迟将可以最小化。

在读 / 写磁头组件（如图 3-19 所示）中，巨磁阻感应器夹在两个磁屏蔽材料之间构成读取部件。这些磁屏蔽材料可以减少磁盘不需要的磁场，这样磁头可以只检测磁头下记录数据位的磁场。在*合并磁头*（merged head）中，第二个磁屏蔽材料同时实现了感应式写磁头的功能。分离读写部件的优势使每个部件可以单独优化。合并

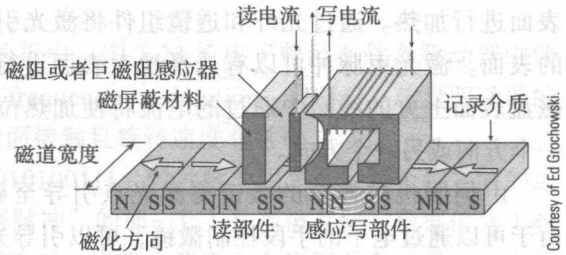


图 3-19　巨磁阻读写磁头的结构

磁头制造更便宜，且由于读写部件之间的距离更小，磁头在驱动器中的性能表现更好。

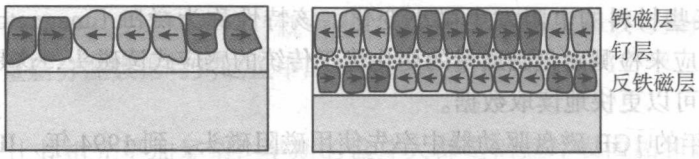
3. 精灵之尘

2001年初，IBM宣布了一项磁盘的突破性技术可以使磁盘面密度增加4倍。IBM类似三明治那样使用3层来存储数据。最上层是存储数据的铁磁材料。最下层是*反铁磁*（antiferromagnetic）层。反铁磁性发生在原子排列与磁场平行但方向相反情况下，即与铁磁性相反。然而，反铁磁性效应非常弱。在这两个层之间就是*精灵之尘*（pixie dust），该层由3个原子厚的元素钌构成。稀有金属钌与铂属于同一族，每年仅生产12t，该层只有3个原子厚可以使少量的钌被使用较长一段时间。

这种三明治组织被称为*反铁磁耦合*（antiferromagnetically-coupled，AFC）介质，它可以将面密度提升到约100Gb/in^2。IBM宣称AFC介质可以避免高密度数据衰减。超薄的钌层使得相邻的两层保持相反的磁化方向。相反的磁化方向使得整个多层结构看上去比实际要薄得多。因此，可以在AFC介质上更容易地记录小而高密度的数据位，且由于介质整体厚度较薄而保持其磁化方向。

早在1990年，IBM的科学家已经发现有钌原子构成的薄层可以在相邻无磁性的铁磁层间产生最强的反向耦合。1997年第一款具有巨磁阻读磁头的磁盘驱动器中就使用了该结构。

图3-20a展示了传统的磁表面，图3-20b展示了钌原子组成精灵之尘的AFC介质。



a）传统介质　　b）三层AFC介质

图3-20　传统介质和AFC介质

4. 光辅助磁头

一种增加位密度的方法是使用*光辅助磁头*（optically assisted head），借助光读/写CD的技术来改善磁头的定位能力。一些含有稀土元素（如，钆）的化合物在室温下具有稳定的磁特性和低居里点。这些材料可以被磁化，就像其他的磁盘涂层物质一样。如果化合物被加热，然后放入磁场中，被加热的区域很容易被磁化为不同的方向。一般的磁化过程通过控制磁场的大小来定位磁化区域。由这些化合物构成的盘片表面，其磁化过程由加热区域的大小控制——即使磁场溢出也无关紧要，因为表面上只有被加热的区域才会被磁化。

图3-21给出了光学辅助写磁头的原理。微小激光对表面进行加热。通过光纤和透镜组件将激光引导至磁盘的表面。激光束脉冲可以有选择地对表面进行加热。在磁盘表面上方的线圈中通过的电流将使加热位置磁化为一个方向或另一个方向。

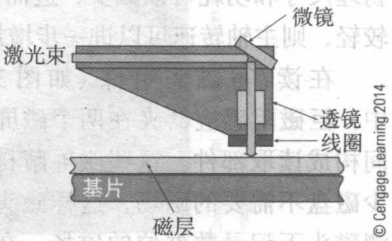


图3-21　光辅助磁头

由伺服机构控制的微镜将激光点引导至磁盘表面。由于可以通过电子的手段控制微镜，可以引导光束在磁盘表面移动（从一条磁道移动至另一条磁道）。该技术可以达到100Gb/in^{2}的密度。

3.3 磁盘上的数据组织

在介绍磁记录过程和磁盘驱动器结构的基础上，下面将介绍数据是如何存储在磁盘表面的。写磁头直接将数据写到磁盘表面。第一代读磁头和写磁头是完全相同的，读写操作使用相同的磁头直到发现磁阻效应。

当读磁头经过磁化表面，磁通量的变化导致线圈中出现感应电流，并且在线圈两端出现电压。线圈两端间的感应电压与磁通的变化率（rate of change）而不是它的绝对值成正比；也就是说，只能检测到磁通密度的变化。

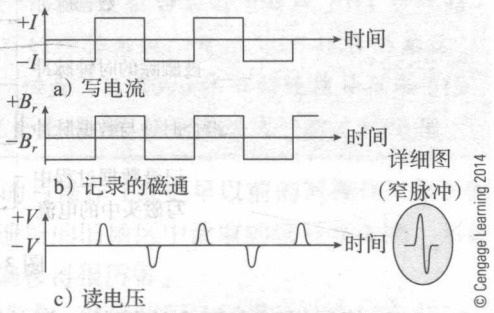


图 3-22 写数据和读数据

图 3-22a 和图 3-22b 给出了写磁头中写电流的情况以及磁表面磁化的结果。图 3-22c 给出了当磁记录表面通过读磁头下方时线圈中的感应电压。图 3-22 中曲线是理想化的，更详细的插图给出了记录脉冲在现实中的样子。

在磁表面不能可靠地存储一长串的 1 和 0，这是因为只有磁通水平的变化才会在磁头中产生信号。如果记录 00000 或 11111，两个序列将会产生相同的结果——没有输出。假设，需要存储串 00011111111110000。读磁头只能检测到两个磁通变化：开始的 0 到 1 以及最后的 1 到 0（在串中以下划线表示）。

巨磁阻磁头可以检测绝对的磁化方向，因为即使固定的磁场也可以使磁阻部件产生可检测的低或高的电阻。然而，在很长一串的 1 和 0 之间不容易看出轮廓。

数字记录方式（包括磁记录和光记录）在记录数据之前进行编码，这是为了避免出现记录的信息很难被读取的情况。特别是它可以避免记录一长串的相同值或者说相同的磁化方向；也就是说，它可以确保记录的磁通量定期地改变状态。编码方式的限制是从磁盘中获取数据波形的需求。大多数的编码方式被称为自同步的（self-clocking），这是因为磁表面上的记录位包含足够的信息让硬件恢复（regenerate）时钟波形，可用于对输入数据进行采样。

设计记录数据的编码方式是一种艺术，因为必须考虑许多相互制约的需求：

- 需要通过减少用来记录每一位数据所需的磁通翻转的数量来提高编码效率（最好的方法是每次翻转记录一位）。
- 需要使 1 和 0 的记录形式尽可能不同，这样可以在存在噪声和其他无关信号的情况下很容易地区分 1 和 0。
- 需要确保在磁通翻转间没有显著的差距来使编码实现自同步。
- 需要避免包含低频部分的模式，因为处理来自读磁头数据的模拟电路不能很好地处理低频。

任意一种数据记录编码都是上述各要求的折中。图 3-23 给出了曾经在软盘驱动器中使用的编码方法，称为改进频率调制（modified frequency modulation，MFM）。软盘驱动器的工作原理与硬盘相同，但其磁头与软盘片磁表面接触且旋转速度非常慢。

图 3-23 给出了需要被记录的位序列；即 010100111。时钟脉冲用来标记每个位之间的边界。编码过程的第一步（图 3-23 标识为“数据脉冲”的那一行）是当被存储的数据位为 1 时产生一个脉冲。如果这些脉冲被直接用于存储数据，当输入流中含有连续两个或两个以上的

0时，由于没有记录数据而出现问题。

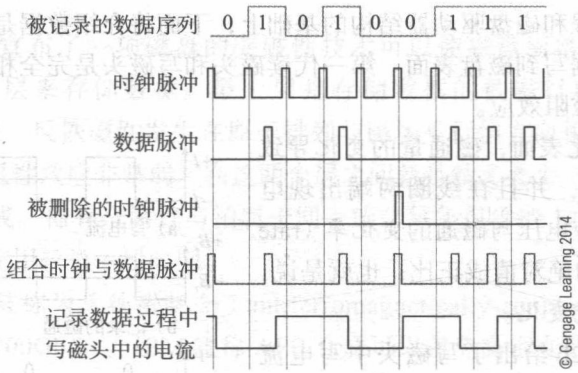


图 3-23 MFM 编码

MFM 编码通过在两个连续的 0 间的边界记录一个 1 来解决缺乏信号的问题。这条规则确保一串 0 也能产生磁通翻转，但是被插入脉冲不能被解释为一个 1，因为它介于信息域的边界，而不是信息域的中间。图 3-23 中最下面一行表示的是写磁头中的电流，它在每个脉冲处改变方向。

改进频率调制编码在 20 世纪 80 年代广泛应用于硬盘，但目前已经被更好的编码方式取代。游程长度受限（Run Length Limited，RLL）码，比 MFM 编码更复杂，但它减少了存储数据所需的磁通翻转。该编码将数据位映射到磁通翻转模式，但限制了 0 序列的最大长度。例如，2,7 RLL 码将编码中 0 序列的长度限制在 2 ～ 7 个。现代的 RLL 编码为 3,9 RLL，将 0 序列的最大长度限制为 9。

今天的磁盘驱动器在记录数据之前采用更加复杂的编码方法，在读出编码后进行译码。特别是，Hitachi 公司在 20 世纪 90 年代率先使用部分响应最大似然（Partial Response Maximum Likelihood，PRML）技术来处理从磁盘中读取的数据。早期的记录技术包括，对数据进行编码，将其写到磁介质上，使用简单的峰值检波器从读磁头中读取脉冲以得到 1 或者 0。然而，高速操作时，由于磁通翻转从磁盘读取的数据是复杂的模拟波形而不是漂亮的方波。此外，如果每位记录得很近，磁通翻转信号的过渡部分将重叠在一起并相互干扰。每位之间的相互作用被称为码间干扰（intersymbol interference），这是自 20 世纪 60 年代一直困扰电话信道调制解调器设计者的问题。PRML 技术涉及从读磁头读回模拟信号，并使用信道特性的知识（即由单脉冲生成波形的知识）来重构原始数据。最大似然这个词表明，译码器选择最有可能生成接收数据的数据记录模式。这种技术也被称为 Viterbi 译码，它在存在噪声的条件下非常有效。

3.3.1 磁道和扇区

前文已经说明，磁盘表面划分为许多磁道。一条磁道被划分为若干扇区，扇区是从磁盘读写数据的最小单位，因为没有办法在磁道上定位某一位然后再修改它。由于没有将磁盘旋转的速度保持为精确的恒定速度（实际生产情况也不允许），有必要一次读取一个数据块。

磁盘驱动器是一种机械装置，写磁头无法精确定位到某个扇区。磁头总有可能向左或者向右偏移，磁头写的扇区可能略微超前或者滞后于需要被覆盖的扇区。

高级文件格式

为了提高磁盘驱动器的数据密度，Samsung 率先倡导高级格式（Advanced Format），它不使用标准的 512 字节扇区而使用 4096 字节的扇区。大的扇区减少了开销。例如，每 4096 个字节而不是每 512 字节需要一个错误校正码。

向 4K 字节扇区高级文件格式发展的趋势，使所有磁盘驱动器制造商从 2011 年开始在其新产品中采用了这一标准。然而，并不是所有的操作系统都可以处理 4KB 的扇区，因此磁盘必须可以模拟 512 字节的扇区，也就是，磁盘具有 4096 字节的物理扇区和 512 字节的逻辑扇区。Windows 7 和 Mac OS 都可以完全支持 4K 扇区高级文件格式的使用。

当从扇区读取数据时，记录的信号中包含先前的写操作（或更早以前的写操作）留下的一小部分数据。从这些由于磁头写操作略有偏移而遗留的旧扇区中读取的信号非常弱，不足以影响到磁盘的操作，它不会成为问题，除非磁头偏移得很厉害。

考虑从读磁头获取模拟信号，构建相应的数据模式。然后使用已知磁头特性的相关知识重组所记录的数据信号，将获得所记录数据的完美副本，而且是没有任何多余信号的副本。如果将从数据中构建的信号与从读磁头中获得的信号相减，就可以得到上述由于磁头偏移写入同一个扇区的先前微小信号。

正前所述，扇区偏移意味着写操作不能根除旧的数据。因为以前写入同一扇区的留下的信号水平太弱，不至于损坏数据，所以实际上不构成问题。然而，FBI 的专家能够读取被删除的文件，他可能不愿意相信，从哥伦比亚订购的包裹真的是妈妈的肉饼秘方。

用户可以订购用来安全地删除数据的程序。它们解决数据不能完全被删除的方法是，不断地将随机的数据写入同一磁道，直到多个写操作已完全毁灭所有先前写的数据。

该操作与恢复（undelete）文件操作不同，文件可以被恢复是因为它仅仅是简单地从目录中删除，其内容仍然存在磁盘上。

图 3-24 给出了磁道的结构。从存储效率考虑，扇区应该尽可能地大。因为每个扇区包含管理专用信息，小的扇区由于浪费了磁盘空间而导致效率低下。从存储效率考虑，扇区也应该尽可能地小。由于扇区是数据存取的最小单位，文件存储的粒度单位都是扇区。如果增加文件的大小，就会导致文件所使用扇区数量的增加。如果最小的扇区（比方说是 4KB），这意味着，平均而言，一个文件的最后一个扇区只有一半存储了有用的信息。如果磁盘有成千上万的文件，浪费空间将非常大。当然，上述两种说法是相互矛盾的。最优扇区大小往往是一种折中。

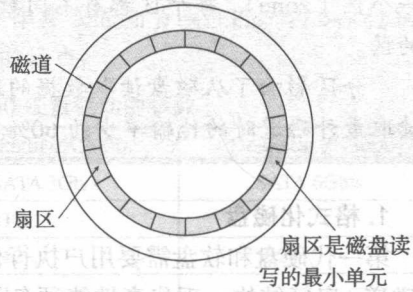


图 3-24 磁道的结构

扇区是保持基本数据单元的数据结构。当磁盘首次使用时，扇区将被写到磁盘表面；也就是说，扇区是软件结构而不是磁盘的物理特性。

图 3-25 给出了磁盘被首次格式化（format）时写到磁盘中磁道的结构（该图为软盘的扇区结构，很容易理解）。直到有了这种结构，磁盘才能用来记录数据。存储数据所需的开销与半导体存储器不同。

间隙中没有有用的信息，但为了给磁头自身与来自读 / 写头的位流同步时间，这些磁道上不同域之间的间隙是必要的。由于磁介质本身不可靠，为了检测已经损坏的数据需要使用

错误检测码。在图 3-25 中，磁道包含索引、地址和数据标记（mark）。这些标记是记录在磁盘上特殊的非法二进制模式（说它们是非法的，是因为它们不符合用于存储数据的数据编码算法）。因此，硬件可以很容易地检测这些标记并使用它们来同步读或写操作。

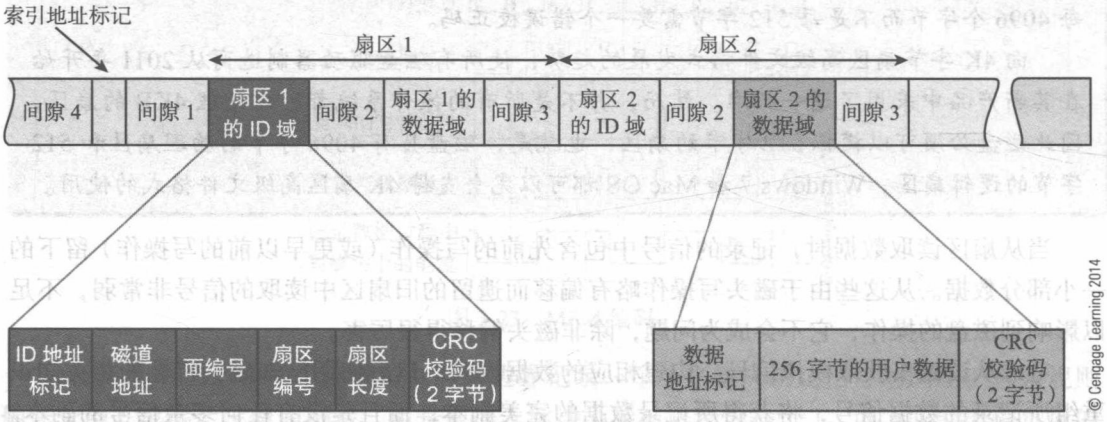


图 3-25 扇区结构实例

分区

磁道的周长是 $\pi \cdot d$，其中 d 是磁道的直径。每个磁道上有 n 个扇区；则扇区的长度是 $\frac{\pi \cdot d}{n}$。记录每一位的尺寸大约为 $\frac{\pi \cdot d}{m \cdot n}$，其中 m 为每个扇区中的位数。由于内外磁道对应的 d 值不同，每位对应的尺寸也不相同。如果最内磁道的位尺寸足够大可以被检测，则最外磁道的位尺寸就会太大了以至于存储效率会大打折扣。

磁盘利用分区（zoning）来处理由于不同磁道长度而导致的问题，相邻磁道被分组为一个区（zone），每个区都有不同数量的扇区。某些今天的磁盘表面划分为 30 个或更多的区。

分区影响了从磁盘读取数据的速度。最内磁道具有较少的扇区，数据传输率可能比读取最外磁道时的传输率少约 60%。

1. 格式化磁盘

第一代硬盘和软盘需要用户执行低级格式化（low level format），这是因为磁盘上没有写上磁道 / 扇区结构。现代高性能硬盘并不是这样，因为它们在出厂前已经进行了格式化。磁盘驱动器可能具有比前面描述的更为复杂的磁道结构。例如，靠近内部的磁道可能由于周长小而具有较少的扇区（分区记录），或者某些扇区因为包含了记录表面有缺陷的地方而被圈起（即看不见）。早期磁盘驱动器比今天的磁盘接口、设备驱动和操作系统出现得更早。那时用户必须创建自己的数据结构，执行格式化，并按照位的级别来读 / 写数据。今天，磁盘驱动器是一个接收来自操作系统命令的黑盒子。用户不必关心数据是如何编码和存储的。

一旦磁盘通过低级格式化具有了磁道和扇区结构，它可以进行高级格式化（high level format）。高级格式化包含操作系统所需的数据结构。因此，不同操作系统的高级格式化结果可能不同。在 PC 上格式化磁盘的操作就是一种高级格式化。

图 3-26 显示了如何从磁盘读取文件。一个文件是由一系列的扇区组成。扇区本身可以

被组织成一个链表（linked list）或目录用来定义属于某一文件的扇区序列。图 3-26 表明，在读文件时可能会出现扇区对应磁道号发生变化，重新寻道导致浪费了大量的时间。当首次创建文件时，它被分配在连续的扇区中。经过一段时间的文件创建和删除，磁盘上可用扇区高度分散，导致文件被严重分散。幸运的是，操作系统可以定期自动或手动整理文件，重组它们的结构以减少寻道时间。

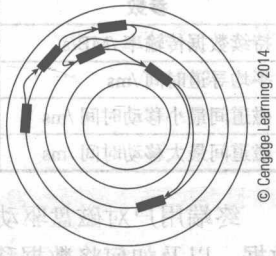


图 3-26 文件可能包含分散在磁盘中的多个扇区

2. 交叉

磁道的扇区按照 0，1，2，…，n-1 进行编号并被称为物理（physical）扇区。假定操作系统需要读取几个扇区。如果首先读取扇区 x，在读取扇区 x 结束及开始读取扇区 x+1 之间，磁盘驱动器电子机构需要做相当多的工作。今天这已经不是一个问题，因为磁盘的旋转速度相对磁盘电子机构中嵌入的高速 16 位处理器来说十分缓慢。扇区间的间隙对任意事务处理来说都是足够的。但历史上并非总是如此。早期的磁盘驱动器内部处理较少，计算机的速度很慢。当处理器准备读取扇区 x+1 时，磁头正在通过该扇区（甚至已经在该扇区之前），因此需要等待扇区 x+1 的起点再次旋转到磁头下方。采用的解决方案是将连续的逻辑扇区映射到交叉的物理扇区上。例如，按照 1:2 交叉的 17 个逻辑扇区在磁盘上的分布可能是：

0，9，1，10，2，11，3，12，4，13，5，14，6，15，7，16，8，...

假设磁盘当前正在读取逻辑扇区 4。下一个逻辑扇区是 5，而接下来的物理扇区是 13。这种安排让计算机在扇区 5 移动在磁头下方之前有时间来执行事务处理。当前的磁盘使用 1:1 交叉，用户不再需要关注交叉的问题。此外，高速缓存和缓冲技术使得交叉成为冗余，因为可以缓冲整个磁道。

3.3.2 磁盘参数和性能

到目前为止，本书还没有讨论磁盘驱动器的任何操作参数的细节或性能。表 3-1 分别给出了一个 120GB、一个 2TB 和一个 3TB 的磁盘驱动器的特点。

表 3-1 120GXB、7K2000 与 7K3000 硬盘驱动的参数

参数	120GXB	Deskstar 7K2000	Deskstar 7K3000
接口	ATA-100	SATA 3Gb/s	SATA 6Gb/s
容量	120GB	2TB	3TB
扇区大小 / 字节	512	512	512
记录区（zone）个数	31	31	
物理磁头	6	10	10
数据盘片	3	5	5
最大面密度 /Gb/in^2	29.7	285	411
最大记录密度 /KB/in	524	1457	
磁道密度 / 磁道 /in	56700	195000	
数据缓存	2MB	32MB	64MB
转速 /rpm	7200	7200	7200
平均延迟 /ms	4.17	4.17	
最大内部数据传输率 /Mb/s	592	1621	1656
最大外部数据传输率 /MB/s	100	300	600

（续）

参数	120GXB	Deskstar 7K2000	Deskstar 7K3000
持续数据传输率 /MB/s	23-48（0-30 区）	134	
平均寻道时间 /ms	8.5	8.2	
磁道间最小移动时间 /ms	1.2	0.6	
磁道间最大移动时间 /ms	15.0		

终端用户对磁盘驱动器的 3 个方面感兴趣：它可以存储多少数据，需要多长时间来访问数据，以及如何将数据移动到主机。磁盘容量（capacity）的计算公式为：面数 × 每面的磁道数 × 每磁道的扇区数 × 每扇区的字节数。

磁盘的访问时间（access time）主要由两部分组成：访问给定磁道的时间，即寻道时间（seek time）；找到磁道后访问给定扇区的时间，即延迟（latency）。延迟很容易计算。假设磁头已经移动到给定磁道，最小的延迟就是零（扇区刚好到达磁头下方）。最大延迟是旋转一圈的时间（磁头刚刚错过了该扇区，必须等待它转回来）。平均来说，延迟为 $1/2t_{rev}$，其中 t_{rev} 为磁盘旋转一周的时间。如果磁盘转速为 7200rpm，该延迟为：

$$1/2 \times 1/(7200 \div 60)=0.00417s=4.17ms$$

平均旋转延迟只能通过增加磁盘的转速来降低。磁盘旋转所需能量与转速的平方成正比。此外，旋转磁盘的应力也与转速的平方成正比。磁盘旋转的最大速度由其能量需求和机械特性决定。构造磁盘材料的自然属性将决定其最大转速。近些年来，磁盘的速度变化相对较小。2004 年，低价的商用硬盘的转速为 5400 转（取代旧的 3600 转标准），高性能磁盘的转速为 7200 转。某些昂贵的硬盘是为 10000 转，而最先进的磁盘为 15000 转。在接下来的 10 年中变化应该不大。

什么是平均寻道时间？如果磁盘有 N 条磁道，磁道间移动时间为 t_{step}。如果磁头在每次寻道后停在磁盘的边缘，寻找给定磁道需要经过磁道的平均值为 $N/2$，则平均寻道时间的计算公式为 $\frac{1}{2} \times N \times t_{step}$。图 3-27a 说明了这种情况。

但是，如果磁头在每次访问后自动移动到所有磁道的中心位置（磁道号为 $N/2$），如图 3-27b 显示。当开始新的寻道操作时，磁头可以向左或向右移动。在这种情况下，寻找给定磁道所需要经过磁道的平均值为 $N/4$，平均寻道时间变为 $\frac{1}{4} \times N \times t_{step}$。

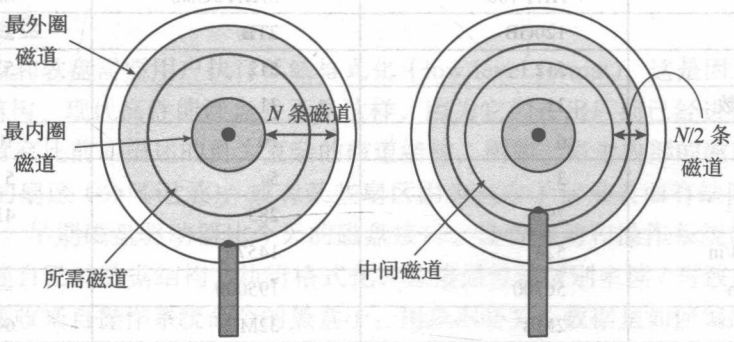


图 3-27 寻道时间和磁头初始位置

在实践中，读写头停留在访问后的位置，所以新的寻道时间有时长有时短。下面计算平

均访问时间。假定磁头停留在磁道 i（i 为 0 ～ $N-1$ 的任意整数），磁头必须一步步定位到给定磁道。磁头移动到每个可能的目的地所需步数如下：

磁头终点	0	1	2	…	$i-1$	i	$i+1$	…	$N-2$	$N-1$
经过磁道数	$i-1$	$i-2$	$i-3$	…	1	0	1	…	$N-i-1$	$N-i$

到每个可能的目的地所需步数的总和为：

$i-1+i-2+\cdots+3+2+1+0+1+2+3+\cdots N-i$

使用等差数列求和公式 $1+2+3+\cdots+k=1/2k(k+1)$，可以得到磁头移动到每个可能的目的地所需步数的总和。

$$\text{所需步数的总和} = 1/2 \times i(i+1)+1/2\ (N-i)(N-i+1)$$
$$= 1/2\ (i^2+i+N^2-Ni+N-Ni+i^2-i)$$
$$= 1/2\ (2i^2+N^2-2Ni+N)$$

平均所需步数为总和除以磁道的数量。也即：

平均经过的步数 $=1/2\ (2i^2+2N^2-2Ni+N)/N$

最后就是利用所有可能的 i 值来计算平均值。也即

$$\frac{1}{N}\sum_{0}^{i=N-1}\frac{(2i^2+2N^2-2Ni+N)}{2N}$$

如果 N 很大，可以将上式简化为：

$$\frac{1}{2N^2}\sum_{0}^{i=N}(2i^2+N^2-2Ni)$$

如果 i 是连续变化的整数，每次寻道需要经过的平均磁道数为 $N/3$。实际上，情况并非这么简单，因为磁头并不是以恒定速率通过磁盘表面。当音圈（voice-coil）电机旋转，磁臂扫过磁盘表面，磁臂会加速和减速，而不是以恒定的速度移动。图 3-28 展示了磁臂从最外圈磁道扫过磁盘表面到达最内圈磁道的情况。

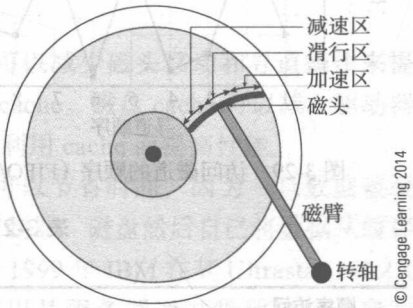


图 3-28 磁头的移动

图 3-28 展示了具有可转动磁臂的磁头运动情况，磁头部分时间在加速，部分时间在磁盘表面以相同的速度滑行，部分时间在减速因为它即将到达目的地。

很难给出磁头在磁道间移动的确切时间。Wilkes 和 Ruemmler[⊖]通过建立磁盘驱动器的模型指出，相比认为磁头是线性运行而言，假定磁头运动为加速、滑行和减速会更好。这 3 段区域的相对长度取决于磁头需要移动的距离。例如，短的移动距离没有滑行阶段，因为磁头的速度还没有加上去。

Wilkes 和 Ruemmler 为转速为 4000rpm 的 HP C2200A 驱动器建立了一个简单的模型。短距离移动 d 个磁道所需的寻道时间为 $3.14+0.5597\sqrt{d}$ms，而 d 比较大时寻道时间为 $10.8+0.012d$ms。可见，当移动的磁道数比较大时寻道时间是线性的，而移动的磁道数比较小时寻道时间与其平方根成正比。这些数字是正是符合人们已知的结论——距离与加速度之间的基本关系为：距离 $=1/2\cdot$ 加速度 $\cdot t^2$。

⊖ Chris Ruemmler and John Wilkes, "An Introduction to Disk Drive Modeling," Computer, March 1994, pp. 17-28.

1. 访问扇区

如果磁盘驱动器顺序读取物理上相邻的扇区，数据传输率为（扇区大小）/（读取一个扇区的时间）。实际上，理想的上限一般不能达到。由于文件创建及修改，文件被分散在不连续的扇区中。此外，计算机在程序执行过程中可能需要访问许多不同类型的数据：代码、子程序库以及帮助文件等。具有多任务操作的系统情况甚至更糟。它们必须应付完全不同程序的请求，而这些程序使用毫不相关的数据。最坏情况下，磁盘在访问扇区间可能要随机寻道（由于采用数据缓存通常不会发生这种情况）。

虽然磁盘数据的测序属于操作系统的领域，这里给出操作系统访问数据的一个短序列来说明体系结构－硬件－软件之间的权衡。

假设操作系统向磁盘驱动器的磁道50、150、32、16、125、8、130、50、60、200发出一系列请求。图3-29画出了根据磁道顺序依次访问——先来先服务（first come first serive，FCFS）时磁头的位置。

操作系统通常为磁盘驱动器提供支持。例如，图3-30中情况与图3-29中基本相同，除了磁道被缓冲，磁头首先按照一个方向运动然后再按照另一个方向运动。表3-2给出了磁臂在这两个阶段中扫过的磁道的数量。可见，对访问顺序重新排序将大大减少磁头的移动。下面为几种典型的磁盘调度算法。在每种情况下，寻道的顺序为10、19、3、14、12和9。

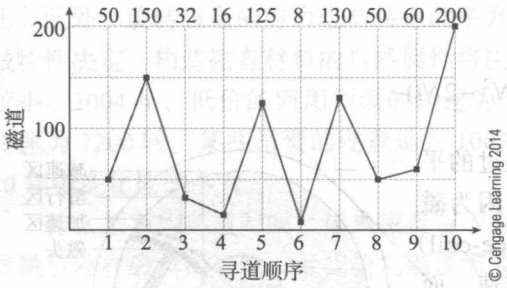


图3-29 访问磁道的顺序（FIFO/FCFS）

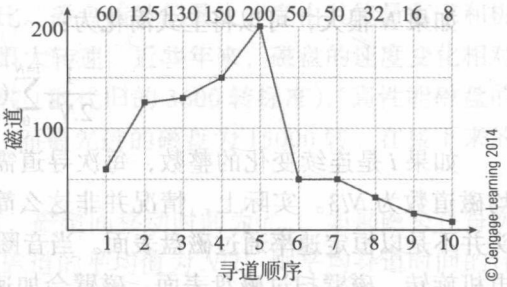


图3-30 非顺序访问（SCAN）

表3-2 访问10个扇区经过的磁道数量

顺序访问											总计
磁道	50	150	32	16	125	8	130	50	60	200	
经过磁道数		100	118	16	109	117	122	80	10	140	812
SCAN											
磁道	60	125	130	150	200	50	50	32	16	8	
经过磁道数		65	5	20	50	150	0	18	16	8	332

FIFO——先进先出（First-in-first-out）也称为先来先服务（first come first serive，FCFS）。该算法处理请求的顺序与其接收的顺序相同。它对所有进程来说是公平的。访问顺序为10、19、3、14、12、9。平均寻道长度为8.2。

SSTF——最短寻道时间优先（shortest seek time first）。下一次寻道的目的地是最接近当前磁头的位置。该算法将磁头需要最小移动的请求作为下一个请求。由于新的请求可能在老的请求之前被服务，这是不公平的。访问顺序为10、12、14、19、9、3。平均寻道长度为5.0。

SCAN——该方法与电梯算法类似，先完成同方向最接近的请求。它先满足所有与当前磁头运动方向相同的请求然后再改变方向。访问顺序为 10、12、14、19、9、3。平均寻道长度为 5.0。

LOOK——这是对 SCAN 简单的修改，它通过软件监测，在当前磁道前方没有请求时改变方向。

C_SCAN——该扫描算法先向一个方向移动，扫过整个磁盘。然后返回到磁盘的另一边重新开始扫描。这是 SCAN 的单向版。访问顺序为 10、12、14、19、3、9。平均寻道长度为 6.2。

FSCAN——该算法旨在处理磁臂粘着（arm stickiness）。它需要两个请求队列。初始情况下，在扫描开始时，所有请求加入一个队列，而另一个队列为空。扫描过程中，所有新的请求放在另外一条队列。这种机制将延缓所有新的请求，直到现有队列中的请求全部被服务。当然，这是一种公平的访问机制。

注意，这些算法的效率会由于寻找扇区而增加的旋转延迟时间而打折扣。

2. 内部磁盘缓存

现在磁盘驱动器都包含一个 RAM 缓冲区，称为磁盘 cache（disk cache），它保存从磁盘读取的数据。这个内部缓存（internal cache）与操作系统设置的磁盘缓存并不相同——尽管它们具有类似的功能。硬盘的内部缓存用来保存最近读操作得到的数据。缓存甚至还可以被编程来预取在不久的将来可能需要数据；也就是说，磁盘可以在给出读取特定扇区的命令之前读取数据。通常，一旦开始读取扇区，该扇区的内容就会被缓存，直到缓冲区已满。一些内部缓存的存储器使用活动分割（activity segmentation）机制，其缓存的数据块的大小是可变的。这允许缓存可以适应正在被执行的访问类型。

通过从 cache 中而不是从磁盘本身获取数据，cache 可以减少磁头移动和寻道操作来提高驱动器的性能。典型的驱动器有 8MB ～ 64MB 的内部 cache。磁盘 cache 可以减少驱动器的平均访问时间；但是，对磁盘碎片的随机访问不能充分利用 cache 而提高性能。

还可以缓存对磁盘的写访问。在写操作时缓存数据可以节省时间，因为一旦数据被缓存，磁盘驱动器就可以将完成（finished）信号返回给主控制器。磁盘然后自己将数据从缓存写入磁盘表面。这种机制被称为写回（write-back）缓存。1999 年 IBM 在其 Ultrastar 18LZX 和 36ZX 磁盘驱动器中实现了完整的写缓存。该特性可以从服务器通过驱动接口开启和关闭。

不幸的是，写回缓存会导致糟糕的意外。如果在缓存数据和写数据操作之间掉电，数据就会丢失。更糟糕的是，已经通知主机进行了写操作，主机并不知道发生了问题。这将导致文件一致性问题，会带来严重的后果。因此写缓存并没有被广泛使用。当系统具有不间断电源时，写缓存才会成为某些磁盘的选项。

3. 传输速率

磁盘用户感兴趣的另一个参数是磁盘存取数据的速率（rate）。该参数很容易计算。如果磁盘的转速为 R 转 /min，每条磁道有 s 个扇区，每个扇区包含 B 位，磁道的容量为 $B \cdot s$ 位。这些位在 $60/R$ 秒内被读（或写）。因此，数据传输速率为：

$$\frac{BsR}{60}\text{b/s}$$

典型的驱动器，例如 Deskstar 7K3000，转速为 7200rpm，读取介质时磁盘最大数据传

送速率为1656Mb/s。而磁盘接口的最大数据传输速率为600MB/s（即4800Mb/s）。

3.3.3 SMART技术

CPU是一种具有大约$10^3 \sim 3 \times 10^9$个晶体管的复杂半导体器件。商业压力迫使制造商在最短的时间内最小化设计、开发和测试周期来进行生产。人们其实并不关注CPU是否能高效地工作，而是关注其是否能正常工作。如果CPU失效，将成为相当大的麻烦。用户将订购一个新的芯片，打开机箱，取出旧的CPU，插入新的。

硬盘是使用前沿技术操作的复杂机电设备。机电系统的可移动部分比其半导体部分的可靠性要低。主要的磁盘制造商开发了SMART技术，用来监控硬盘驱动器的性能，可以预测失效的可能性，从而为用户提供可能出现失效的预警。SMART是自监测、分析和报告技术（Self-Monitor Analysis and Reporting Technolog）[⊖]的简称。

IBM和Compaq公司都独立开发了自己的SAMRT的初级形式，然后将其专业知识和技术整合在一起形成了SMART。IBM的预测故障分析（predictive-failure analysis，PFA）使用磁头飞行高度等参数的测量值来指示可能的失效。Compaq与Seagate和Quantum一起开发了IntelliSafe技术来测量驱动参数，并与预先设定的阈值进行比较。如果超过阈值，就会将状态消息发送到主机。每个驱动器的参数和阈值都不相同，但将状态发送给主机的方式对所有系统都是基本一致的。SMART技术的实现（比如IntelliSafe）因系统而异，这是因为驱动器的体系结构不断创新，某些驱动器比其他驱动器更加适合关键任务（mission critical）的应用。

某些电子故障是突然出现和无法预测的。机械问题相对来说可以预测。在日常生活中，汽车驾驶员可以观察汽车轮胎的松弛和刚度状况以及油位情况。所有这些参数给出了关于汽车状态的有价值的信息。磁盘驱动器的某些参数也可以用来指示可能的失效。

SMART参数

检测硬盘驱动器性能的SMART技术使用了几个与驱动器操作有关的参数。最终，这些参数都是在磁盘老化后用来检测迟钝情况的。下列是其中的一些参数。

磁头飞行高度——磁盘驱动器最关键的参数之一是读/写磁头在磁盘表面上的高度。该值确实非常小，大约为微英寸（μin）量级。磁头-表面高度由磁盘旋转时表面的空气流动控制。过低的间隙使磁头具有撞击旋转磁盘表面并破坏磁盘表面的危险。过高的间距使从磁表面读出的信号太弱，来自多个位的信号有可能会相互影响，写信号的强度可能不足以对表面进行磁化。这是一个可以通过测量读磁头信号的强度而获得的简单参数。

数据吞吐率——数据吞吐率用来表示磁盘驱动器提供数据的速率。一些机械问题可以导致平均数据吞吐率的减少。

转轴加速时间——转轴加速时间指的是磁盘驱动器达到其工作速度所花费的时间。如果该时间增加，它表明控制电路可能出现了问题，或者更有可能的是转轴驱动电机已经磨损。

转轴重试次数——如果驱动器在一个给定的时间内没有达到其工作的速度，将会重

⊖ "Get SMART for reliability," Paper TP-67D, July 1999, Seagate Technology, Scotts Valley, CA.

新尝试。转轴重试次数用来记录转轴达到规定转速所用的次数。这个参数用来指示转轴电机的问题或电源供电的问题。

重新分配扇区数量——如果发生读写校验错误（有可能是由于磁表面存在缺陷），将重新分配一个扇区并将数据转移到这个新的区域。因此，坏扇区将被自动映射到好的扇区（这将导致数据被分散和数据传输率的下降）。驱动器将使重映射（重新分配）的扇区数量保持在一个范围内，此参数值的增加表明可能存在可靠性的问题。

寻道出错率——寻道操作使磁头移动至所需的磁道。如果寻道失败，必须再次尝试。寻道通常由于热问题（使磁臂和磁盘膨胀）导致错误，发热问题可能是由于定位机构或者磁盘表面受损而引起的摩擦导致。如果寻道出错率增加，失效的可能性也会增加。

寻道时间——寻道时间用来指示定位给定磁道所需的时间。寻道时间的增加表明机械部分可能出现问题。

磁盘磁头校准重试——为保证正确的寻道，磁盘会进行自动校准过程。如果该过程失败，将重复该过程（即重新校准）。如果重新校准请求过于频繁，很可能是由于机械故障或电源问题（甚至是读/写头本身的问题）。

Seagate 在其技术报告《Get SMART for reliability》(编号 TP-67D）中讨论了 SMART 技术以及增强技术。该文特别指出，存在疑似问题而返回 Seagate 维修的驱动器中有 40% 被证明是好的。这些驱动器被返回是因为主机系统中的其他问题（例如，软件错误或病毒的影响）。

1999 年，Seagate 引入了驱动器自测试（drive-self test，DST）技术，它为预测磁盘失效提供了更为积极主动的方法。它在驱动器的固件中嵌入了各种测试。例如，快速测试（quick test）花 2min 读取磁盘第一个 1.5GB 的内容；扩展测试（extended test）通过完整地扫描存储介质进行更彻底的检查。扩展 SMART 技术将驱动器最近报告的错误记录并保存下来。该日志也可以用作诊断。

温度对磁盘可靠性的影响

电子元器件的可靠性高度依赖于温度。电子和机械部件（如转轴电机和传动装置轴承）的可靠性都会随温度上升而降低。磁盘驱动器在高温下长时间工作将极大地降低其寿命。在一篇有意思的文章《IBM's Drive Temperture Indocator Processor (Drive-TIP) helps ensure high drive reliability》[⊖]中讨论了温度和磁盘可靠性之间的关系。

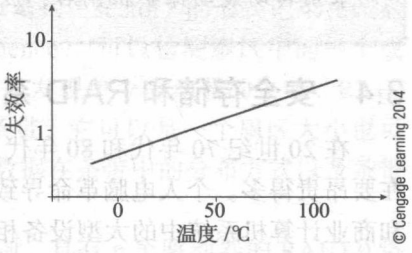


图 3-31 温度与磁盘可靠性

图 3-31 显示了温度与系统失效率之间的关系。失效率是温度的指数函数，通常由 Arrhenius 公式计算：失效率 $=A\exp(-E/kT)$，其中，A 和 E 是常量，k 是 Boltzman 常数，T 是绝对温度。通常，温度每升高 1℃磁盘驱动器失效率增加约 3%。

高温可以导致一些失效。例如，磁盘或磁臂的热膨胀会导致数据写在磁道外部，从而使相邻磁道的数据出现错误。较高的温度会使主轴电机和音圈电机中的润滑剂因汽化而泄漏，从而导致磁头阻塞甚至损坏。

⊖ Gary Herbst, "IBM's Drive Temperture Indocator Processor (Drive-TIP) helps ensure high drive reliability," IBM Storage System Division, 5600 Cottle Road, San Jose, CA, 1997.

热量来自于计算机中的芯片和磁盘驱动器，散热需要通过风扇或自然的对流实现。风扇失效或排气口堵塞会使温度上升进而导致磁盘失效。IBM 在其某些磁盘驱动器中提供了温度检测功能以进行预警。这些驱动器可以自动监控温度，当超过其最大允许的温度时向驱动器控制器报警。在实践中，只监控两种温度：用户选择的最低温度，以及驱动器构件允许的最高温度 65℃。

驱动器启动后就开始每 25min 读取一次温度值。当温度超过第一个温度值（用户设定的最低温度）时，采样周期从 25min 一次变为 15min 一次。此时，在永久驱动错误日志中会记录一个条目，包括温度和已通电小时数（power-on hours，POH）。只要温度高于第一个温度值，它将不断创建日志条目。如果温度超过 65℃，采样周期从 15min 一次变为 10min 一次。

IBM 的驱动器温度监控可以与其开发的 SMART 标准一起监控和预测设备的性能和可靠性。如果发出警告，可以采取保护数据的一些措施以免造成难以弥补的灾难。

摇晃和震动

所有的机械系统对物理上的干扰都十分敏感，如振动。可以想象，使读/写头保持在 50nm $\left(\text{约是人类头发的直径 100μm 的}\frac{1}{2000}\right)$ 的磁道中央并以 7200rpm 的速度旋转是十分不易的。磁盘驱动器使用反馈控制将磁头保持在磁道中央；也就是说，使用磁头没有正确定位这个误差信号来移动磁臂用以减小误差。

当驱动器受到外部振动，磁头偏移当前位置，将产生更大的误差信号，利用该信号可以使磁头返回正确的位置。不幸的是，这是一种事后（post hoc）解决方案；在打扰发生之后才进行修正。Hitachi 白皮书《Rotational Vibration Safeguard》中描述了一种用于减少振动产生影响的新技术。它在磁盘驱动器的电路板上装有两个振动传感器（重力传感器和加速度计）来检测运动。来自这些传感器的信号将控制磁头传动装置在渐渐偏离磁道之前移回正确的位置。在实际测试中，震动将使磁盘性能减少到原来的 30% 左右，而采用转动震动保护机制后，应用的性能提高到 90%。

3.4 安全存储和 RAID 系统

在 20 世纪 70 年代和 80 年代，硬盘的容量比现在小很多，每兆字节花费的美元也比现在要昂贵得多。个人电脑革命导致了中小尺寸的硬盘成本快速下降——这种下降最初与专业和商业计算机系统中的大型设备相应的价格下降并不匹配。

失效概率

假设驱动器在给定时间内的失效概率为 $1/n$。此处称之为 p，如果在 100000h 内出现 1 次失效，p 就是 0.00001。如果驱动器阵列中有 m 个驱动器，整个阵列的失效概率是多少？

考虑阵列中一个驱动器失效，其余 $m-1$ 个驱动器不失效的情况。单个驱动器没有失效的概率是 $1-p$。由于失效相互独立，两个驱动器都没有失效的概率是 $(1-p)(1-p)$。故，m-1 个驱动器都没有失效的概率是 $(1-p)^{m-1}$。此时，某个驱动器失效的概率是 $p(1-p)^{m-1}$。然而，由于 m 个驱动器中的任意一个都可能失效，因此整个阵列中只有一个驱动器失效的概率是 $m \cdot p \cdot (1-p)^{m-1}$。

将该概率扩展为两个驱动器失效的情况。此时两个驱动器失效，其余 m-2 个不失效。此外，由于可以从 m 个驱动器中任选两个，其可能性为 C_m^2，也就是 $\frac{m!}{2!(m-2)!}$。因此整个阵列中有两个驱动器失效的概率是 $C_m^2 p^2[1-(1-p)^{m-2}]$。

在 20 世纪 80 年代后期，研究人员意识到低成本的磁盘驱动器可按照新方法使用。1987 年，UC Berkely 的 Patterson、Gibson 和 Katz 发表了一篇论文《A case for redundant arrays of inexpensive disks (RAID)》[⊖]，提出了一种在 PC 中构成廉价磁盘系统的方法。廉价磁盘阵列（array of inexpensive disk）意味着使用当前的商用磁盘驱动器构成了一种结构，而冗余（redundant）意味着一定程度的容错；也就是说，单个驱动器故障不会使整个系统失效。

RAID 的概念快速从研究实验室中传播出去，到 20 世纪 90 年代中期，RAID 系统开始在个人计算机杂志上打广告。今天，大多数 PC 机主板都支持 RAID。Patterson 等人提出了将驱动器组织成 RAID 0、RAID 1 等结构的方法。各种 RAID 级别提供不同的功能，某些强调速度，某些强调可靠性。实际上，并非所有的 RAID 今天都在广泛使用，有些混合系统实现了将两种 RAID 级别相结合的特性。

磁盘驱动器本质上是串行存储设备。在 RAID 系统中，多个磁盘驱动器可以并行操作，单个文件的不同部分可以分布在不同的驱动器中。RAID 阵列可以用来提高磁盘系统的性能（performance）或改善可靠性（reliability）。通过在多个磁盘中复制数据，系统将感受不到单个磁盘的失效。RAID 系统对于数据安全至关重要的应用场合（例如银行）是十分重要的。

n 个磁盘构成驱动器的可靠性只有单个磁盘的 $1/n$，这是因为 n 个磁盘都会失效。同样，四引擎飞机比单引擎飞机更容易出现引擎故障。但是，飞机有 3 个引擎还可以继续飞行，而单引擎飞机的引擎故障将是灾难性的。RAID 系统的作用也一样。如果数据分布于磁盘阵列中，单个磁盘的故障不会导致系统崩溃。这种说法对 RAID 0（0 级 RAID）不成立，它只是一个特例。

为了了解 RAID，需要回忆两个概念。首先，数据是以扇区（sector）的形式记录在磁盘上的。第二，每个扇区都有帧校验序列（frame check sequence），可以检测扇区中的一个或多个错误。这种检测错误的能力意味着 RAID 阵列可以快速发现单个驱动器的失效。RAID 技术的关键概念是条带（strip）。整个磁盘空间分为若干条带，它可以是一个扇区大小也可以是几兆字节。这些条带以交叉方式分布在多个磁盘中。数据在条带中的分布方式以及条带在磁盘中的分布方式决定了 RAID 的级别。

RAID 0 是唯一不使用冗余来提供额外安全保障的级别。具有 n 个驱动器的 RAID 0 将数据划分为 n 个条带并保存在 n 个驱动器中。图 3-32 展示了具有 4 个驱动器的 RAID 0。每个磁盘驱动器的大小相同，否则 RAID 阵列的有效容量默认以容量最小磁盘的大小计算。

RAID 0 阵列的优点是高吞吐量。n 个驱动器的容量是单个驱动器的 n 倍，由于读操作可以并行执行因此速度较高。RAID 控制器可以通过硬件或软件来实现。然而，正如前面所说的，常见的 RAID 控制器内置于 PC 的主板中，用户可以做的只是插入磁盘驱动器和配置 BIOS。

⊖ D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," SIGMOD'88 Proceedings of the 1988ACM SIGMOD International Conference on Management of Data, Volume 17 Issue 3, June 1988.

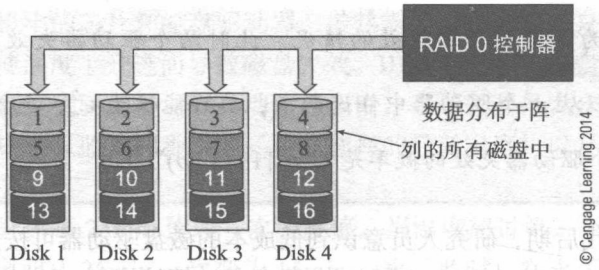


图 3-32 RAID 0 的条带

BIOS

PC 的基本输入 / 输出系统（basic input/output system，BIOS）驻留在 Flash 存储器中，作为加电时的引导程序。BIOS 的功能是从硬盘、闪存或光盘中加载操作系统。BIOS 也可以用来定义系统参数，如 CPU 时钟频率、CPU 电压以及 DRAM 时序等。传统 BIOS 正在消失，逐渐被统一可扩展固件接口（unified extensible firmware interface，UEFI）取代。BIOS 变为 UEFI 是必要的，这是因为 BIOS 是在 16 位处理器和小容量磁盘的时代设计的。UEFI 设计用来支持 32 位和 64 位处理器以及超过 3TB 容量的驱动器。2011 年，UEFI 开始出现在主流主板中。

具有 n 个驱动器的 RAID 0 阵列的容量是单个驱动器容量的 n 倍；也就是说，由于冗余并没有损失容量，存储效率是 100%。然而，由于没有容错，阵列中任意驱动器的损坏都会导致所有数据失效。RAID 0 的读写性能很出色。

因为一个磁盘的失效会导致整个系统失效，RAID 0 阵列只有在经常备份且数据失效时可以从备份中获取时才有意义。在这种情况下，RAID 0 内在的并行读写机制使其在磁盘空间利用率和速度上都是高效的。

1. RAID 1

图 3-33 展示了一个 RAID 1（1 级 RAID）阵列，由于它将条带的副本复制到多个驱动器（这里使用两个驱动器并行操作）中，因此也被称为镜像（mirroring）。由于可以删除一个驱动器而不丢失数据，因此数据的安全性得到加强。RAID 1 系统不仅增加了数据安全性，它还提高了访问性能。假设需要访问给定的条带，在访问过程中，系统将可以从提供数据的第一个磁盘中读取该条带。

RAID 1 的写时间由两个写操作中较长的操作决定。幸运的是，大多数的访问为读操作，而不是写操作。此外，可以缓存写操作，允许磁盘在空闲的时候完成写操作。RAID 1 阵列由于复制了数据所以昂贵，但它可以通过几百美元的大容量高速磁盘提供性价比较高的安全性。

由两个磁盘构成的 RAID 1 系统由于数据是简单复制的，所以效率是 50%。复制提供了出色的容错性能。如果一个驱动器失效，系统可以继续正常工作。所要做的是去除失效的驱动器，安装一个新的，然后重建丢失的数据。大多数 RAID 1 控制器支持对失效驱动器的自动重建。

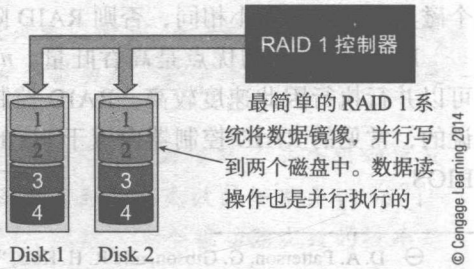


图 3-33 RAID 1

称为 RAID 0+1 或 RAID 0/1 的混合系统通过提供快速数据访问和驱动器失效保护，结合了 RAID 0 和 RAID 1 的特性。图 3-34 显示了两套、3 个驱动器构成的系统。条带写在驱动器 1、2 和 3 中，提供 RAID 0 的服务。然而，驱动器 1、2 和 3 镜像为 4、5 和 6，这种安排提供了 RAID 1 系统的安全性。这样的安排是 RAID 中最昂贵的形式。

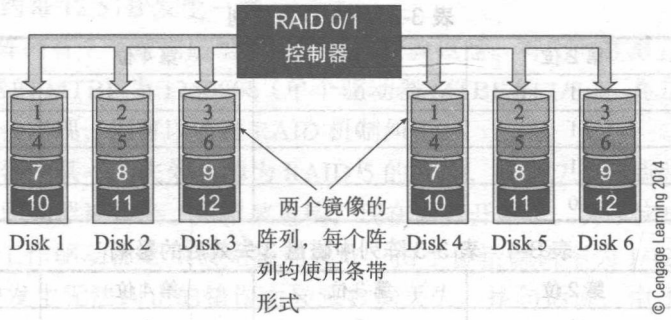


图 3-34 RAID 0/1

2. RAID 2 和 RAID 3

RAID 2 到 RAID 6 都像 RAID 0 那样以条带形式存储数据，但它们并不像 RAID 1 那样完全复制数据。换句话说，RAID 2 ～ RAID 6 介于两种极端情况 RAID 0 级（没有冗余）和 RAID 1（复制冗余）之间。

汉明码

汉明码（Hamming code）是错误检测码和纠错码系列中最简单的成员。汉明码使用多个冗余位，每一位都需要保护数据字各数据位的奇偶校验位。如果发生一位错误，数据字的一个或多个奇偶校验位将不再正确，可以用它们来定位错误并纠正错误。

现在的错误检测码和纠错码远比汉明码更强大，可以纠正多个错误。另一方面，汉明码非常容易实现，适合可能只有一个磁盘失效的环境中。

RAID 2 和 RAID 3 采用多个同步的磁盘驱动器；也就是说，各磁盘转轴同步工作，以便读 / 写磁头同时通过阵列中的每一块磁盘的第 i 个扇区。RAID 2 和 RAID 3 阵列提供真正的并行访问，典型情况是，一个字节被并行写入每个磁盘。RAID 2 和 RAID 3 的区别是，RAID 2 使用汉明码来进行错误检测和校正，而 RAID 3 只使用一个简单的校验位来检测错误。RAID 3 的奇偶校验数据存储在一个磁盘上，而 RAID 2 使用的汉明码可能分布在多个驱动器中。图 3-35 给出了 RAID 3 的概念图，它也被称为位交叉奇偶校验（bit-interleaved parity）。

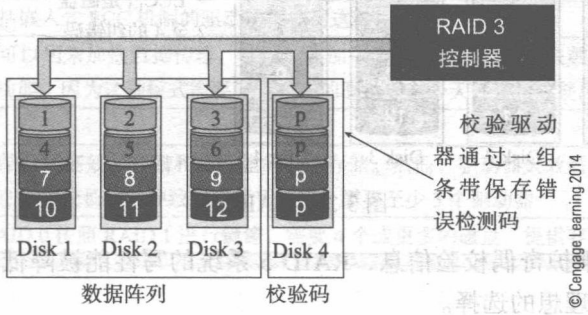


图 3-35 RAID 3

一位奇偶校验码通常不能用于纠正错误。但它可以在 RAID 3 阵列中纠错。假设一个磁盘驱动器失效，因此在失效磁盘上记录的条带将丢失。然而，奇偶校验磁盘上记录的条带可以用来重建丢失的数据。表 3-3 给出了具有 4 个数据磁盘和一个奇偶校验磁盘的 RAID 3 阵列。如果磁盘 3 失败，会出现表 3-4 的情况。

表 3-3 RAID 3 阵列

第 1 位	第 2 位	第 3 位	第 4 位	校验磁盘
0	1	0	0	1
1	1	0	0	0
0	1	1	1	1
1	0	1	0	0

© Cengage Learning 2014

表 3-4 表 3-3 阵列中磁盘 3 失效后的影响

第 1 位	第 2 位	第 3 位	第 4 位	校验磁盘
0	1	?	0	1
1	1	?	0	0
0	1	?	1	1
1	0	?	0	0

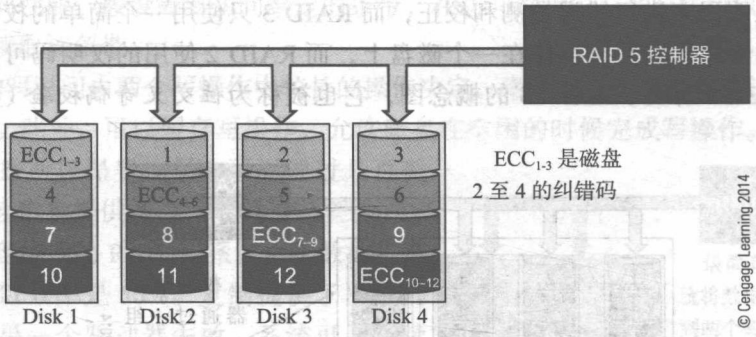
© Cengage Learning 2014

因为知道每一行的奇偶校验位，可以重新计算丢失的数据。例如，第一行的各位是 0、1、?、0 和 1。由于校验位是奇数，因此数据位中必须有奇数个 1。因此，丢失的位必然为 0。

3. RAID 4 和 RAID 5

RAID 4 和 RAID 5 与 RAID 2 和 RAID 3 相似。然而，RAID 4 和 RAID 5 中，单个磁盘不再同步而是彼此独立地操作。其条带远比 RAID 2 和 RAID 3 的要大。RAID 4（块交叉奇偶校验，block interleaved parity）中，奇偶校验条带被存储在一个磁盘上，而在 RAID 5 中，奇偶校验条带被交错存储在阵列中的所有磁盘上。当更改数据时可以通过修改相应的奇偶校验位来高效地更新 RAID 5 中的奇偶校验信息。

图 3-36 所示的 RAID 5 是一种流行的配置，提供了条带存储以及错误恢复所需的奇偶校验。奇偶校验块分布在阵列中的各个驱动器中，使访问负载在驱动器中更加平衡。RAID 5 阵列至少需要 3 个驱动器。



© Cengage Learning 2014

图 3-36 RAID 5

由于需要时间来维护奇偶校验信息，RAID 5 系统的写性能被降低，所以如果需要最高的性能，RAID 5 不是理想的选择。

4. RAID 5 失效的例子

Permabit Technology 公司在其白皮书[⊖]中提供了一个有关 RAID 失效成本的有趣例子。他们考虑的 RAID 5 系统由 500GB 的 SATA 驱动器构成，驱动器的平均无故障间隔时间（Mean Time Between Failure，MTBF）为 1 000 000h，位错误概率为 $1/10^{14}$ 位或者 1/12.5TB。这意味着读失效大约每 12.5TB 发生一次。

假设 RAID 5 阵列有 7 个数据磁盘和一个奇偶校验磁盘。有用的数据容量为 7 × 500GB= 3.5TB。8 个驱动器的 MTBF 为 125000h（单个驱动器 MTBF 的 1/8）。在正常操作中，无法纠正的读操作可能会出现，但可以通过 RAID 机制纠正。

假设 8 个驱动器中某一个失效。因为 RAID 5 的属性，其余 7 个磁盘将继续以降级模式运行，直到失效的驱动器被替换、数据被重构。现在噩梦开始了。为了完成数据重构，包含 3.5TB 数据的 7 个工作驱动器必须被完整地读取。由于位错误概率为 1/12.5TB，有 28% 的可能在重建过程中发生无法纠正的错误而导致数据丢失。换句话说，由于单个驱动器失效而重构所需巨量数据，缺乏进一步的冗余意味着几乎有 28% 的可能是所有数据都无法恢复。显然，RAID 并不能完全防止错误。

5. RAID 6

RAID 6 是 RAID 5 的扩展，其组织除了一个变化外与图 3-36 基本相同。它也像 RAID 5 阵列那样存储纠错信息，而且每个驱动器中存储了第二个校验块。RAID 6 与 RAID 5 的特性类似。然而，额外的奇偶校验信息使其在一个驱动器失效后的重建过程中可以纠正错误。

RAID 阵列对失效的处理

当 RAID 阵列中一个硬盘失效时，必须更换该硬盘。RAID 控制器可以设计成热切换（hot swappable）的；也就是说，可以在不关闭电源和重启的情况下取出失效的驱动器并插入一个新的进行替换。更换驱动器后，新的驱动器必须为阵列进行配置。在 RAID 术语中，该操作称为重建（rebuilding）。重建操作在正常操作继续执行时发生（这也是 RAID 系统的全部意义）。

重建 RAID 1 系统中的驱动器相对容易，因为所要做的就是从一个工作驱动器向新的镜像驱动器复制数据。重建 RAID 5 阵列中的数据需要更多的时间，因为需要从其他驱动器中读取合适的条带来合成所有的数据，然后再执行数据的异或操作。下表总结了各种 RAID 的特性。

RAID 级别	特性
RAID 0	最快和最有效的安排。不提供容错机制。需要至少两个驱动器
RAID 1	数据镜像（复制）。性能攸关、容错环境下的最好选择。需要至少两个驱动器
RAID 2	当今已包括嵌入式 ECC 机制的驱动器不使用这种模式
RAID 3	这种模式可以用来加快数据传输，并通过增加一个包含用来重建丢失数据的纠错信息的驱动器来提供容错功能。因为这种模式需要转轴同步的驱动器，今天已经很少使用。至少需要 3 个磁盘
RAID 4	很少使用
RAID 5	这种模式结合了高效、容错和高性能的数据存储。然而，驱动器失效后的性能很差，由于重建奇偶校验信息需要较长时间导致重建过程缓慢。需要至少 3 个驱动器
RAID 0+1	对整个 RAID 0 按照 RAID 1 进行镜像，需要 4 个或更多的磁盘。提供容错和改进的性能

⊖ "RAIN-EC: Permabit's Revolutionary New Data Protection for Grid Archive," January 2008.

（续）

RAID 级别	特性
RAID 1+0	对每个磁盘先按照 RAID 1 进行镜像，然后再构成 RAID 0。与 RAID 0+1 类似，但性能更好
RAID 5+1	对整个 RAID 5 按照 RAID 1 进行镜像。至少需要 3 个磁盘

JBOD

存储世界中的另一种记忆手段是磁盘簇（just a bunch of disks，JBOD）。JBOD 不是 RAID 的近亲；它更像远房表亲（就像某个作家评论的那样）。JBOD 技术允许将多个驱动器组合为一个卷（与磁盘分区完全相反）。

JBOD 技术的优点是，允许用户将多个磁盘（例如旧系统中不再使用的磁盘）组合创建一个新的大容量磁盘。与 RAID 技术不同，JBOD 并不提高性能也不提高可靠性。

3.5 固态盘

通过前面的介绍可见，机电磁盘驱动器是技术的奇迹，3.5 英寸形状的磁盘容量可以达到 4TB。然而，它的生存空间受到引入固态盘（solid-state drive，SSD）的限制。SSD 可以用来模拟硬盘。固态硬盘使用半导体闪存技术来存储数据，其电子接口与硬盘接口物理上兼容；也就是说，可以将 SSD 插入硬盘的 SATA 插座。串行高级技术附件（Serial Advanced Technology Attachment，SATA）是从磁盘驱动器到主机控制器的一种低成本、高速串行存储设备接口。

SSD 的策略

商业组织开始热衷于“绿色”问题，因为这样做有利可图的。例如，酒店要求你重复使用毛巾以节约用水，这样可以保护环境，也可以节约酒店的能源 / 水资源。

同样，驱动器制造商打着绿色的旗号大力推广固态盘。虽然这种推广是逐利的，但固态盘确实是绿色的。因为 SSD 没有移动部件（特别是没有旋转的盘片），其能耗本身就低于硬盘驱动器。此外，SSD 可靠性的增加意味着它们的更换频率更低，可以节省制造替换驱动器导致的能源消耗。

相比机电磁盘，SSD 具有巨大的优势——最重要的是其更高的性能、更低的能耗、重量更轻，以及更能容忍震动。2010 年，SSD 开始在高端笔记本电脑（2.5 英寸）和专业高端应用（3.5 英寸）中初见端倪。2012 年，SSD 成为高级笔记本电脑的标配。SSD 的局限性是双重的：它们的成本相对较高且容量相对较小。

固态盘由前面章节中介绍的闪存技术构成；正是由于近年来闪存的价格降低到一定程度，才使得大容量（128GB）固态盘在经济上成为可能之选。

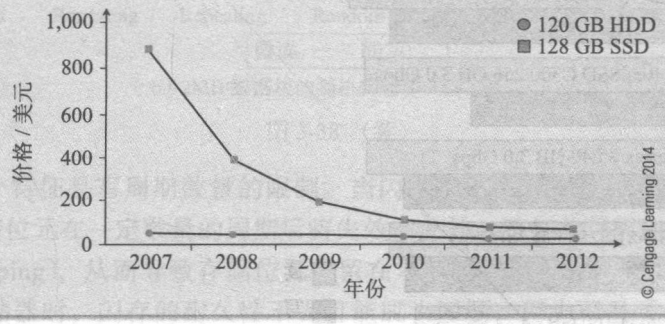
由于固态盘没有移动部件，它们是真正的随机访问设备。固态盘没有旋转延迟时间和寻道时间。因此，与硬盘碎片有关的问题自然就消失了。当文件分布于整个存储空间时，不需要定期地整理 SSD。早在 2007 年，一份来自 IDC[⊖] 的白皮书指明了基于 SSD 的笔记本潜在的收益。

⊖ J. Janukowicz and D. Reinsel, “Evaluating the SSD total cost of ownership,” IDC, Novermber 2007.

- PC 部署的 IT 劳动力收益 2.4%
- 外包维修的可靠性收益 1.4%
- 维修的可靠性收益 7.5%
- 针对硬盘丢失而消耗用户生产力的可靠性收益 17.2%
- 能力增强收益 16.9%
- 用户生产力收益 54.5%

SSD 的价格

像所有新技术一样，SSD 刚出现时与硬盘相比有巨大的差价。但随后，其价格迅速下降。



当然，这些数字适用于特定的企业环境：在特定的时刻，并且基于许多假设。然而，其原理是有效的。固态盘取代硬盘会带来利益，而获取的利益会导致新一代固态盘的发展[⊖]。

固态盘的特点

标题中“特点（feature）”这个词颇有讽刺意味，类似那句话“这是一个特点，而不是污点”。磁盘的读、写和擦除都要花费时间。当然，这是必需的，因为磁盘表面在读 / 写头下方通过，可以读取或写入的时间是固定的，且由系统的机械部分决定。固态存储器使用不同的读和写机制。例如，读取闪存单元，需要向硅通道注入电流，然后检测浮栅上存储的电荷是否允许电流通过。向位元中写数据时，需要向浮栅上注入电荷。该过程可能需要多个步骤，因为有些系统分几步注入电荷并测试数据是否被存储（在存储足够的电荷前可能需要执行多次写）。清除位元的过程更为复杂，因为电荷必须被驱除出浮栅。

航空公司间接促进固态盘的发展

有时，人们不太去注意或谈论技术发展的原因。我以前的一位同事去一家涉及计算、网络和数据分布的跨国组织工作。他问到计算机技术的驱动力是什么。我给出了一般的回答。然后他告诉我一个重要的原因是连锁酒店房间里提供的电影服务。在大酒店里允许几百人进行视频点播将消耗大量带宽。

同样，我决定购买一台带 SSD 的笔记本是因为在某些航班上限制了手提行李的重量。超便携笔记本电脑通常不提供 DVD 驱动器以节省重量 / 尺寸。然而，一些高端笔记本电脑提供 DVD 和 SSD 驱动器。

⊖ 固态硬盘也有负面影响。比如如果一个小偷闯入会议厅，正好代表们出去吃午饭，留下了他们的笔记本电脑，小偷可能会带着有 SSD 的笔记本逃跑。

由于不同的读、写和擦除机制，SSD 的读写时间不相等。一般来说，SSD 的读取时间比写时间要快很多。图 3-37 展示了几种 SSD 在随机读或写 2MB 数据时的性能。水平长度表示数据传输率，单位为 MB/s。注意 SSD 不同模型之间的显著区别[⊖]。

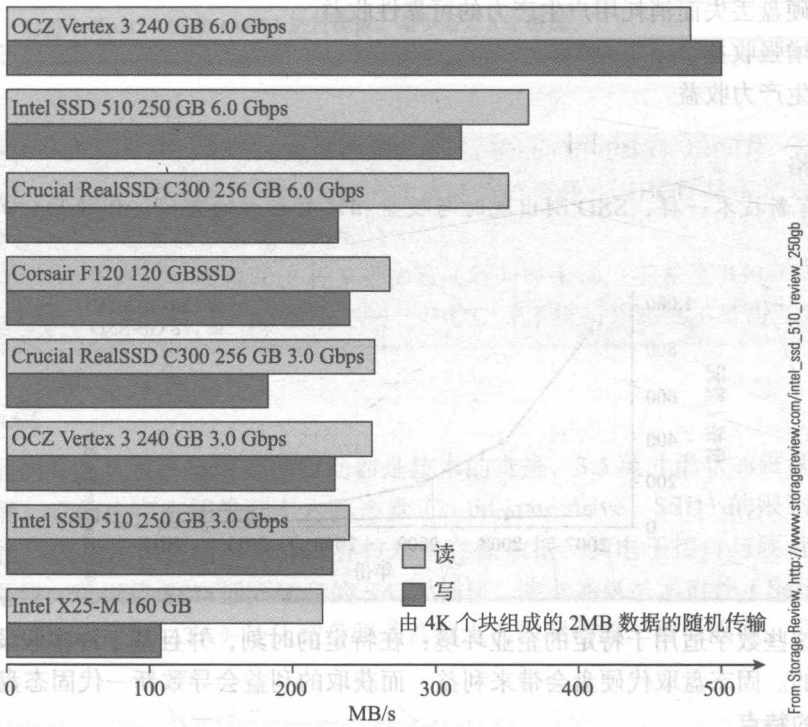
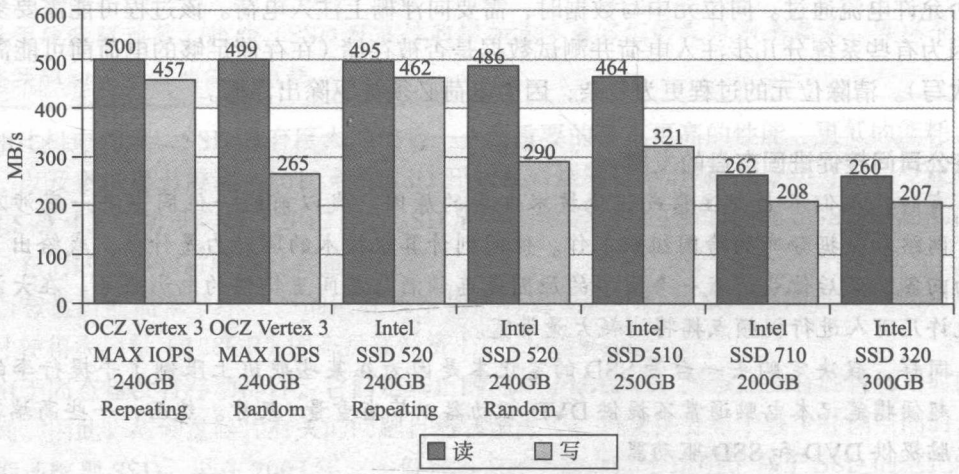


图 3-37 SSD 2MB 随机传输（4K 扇区）

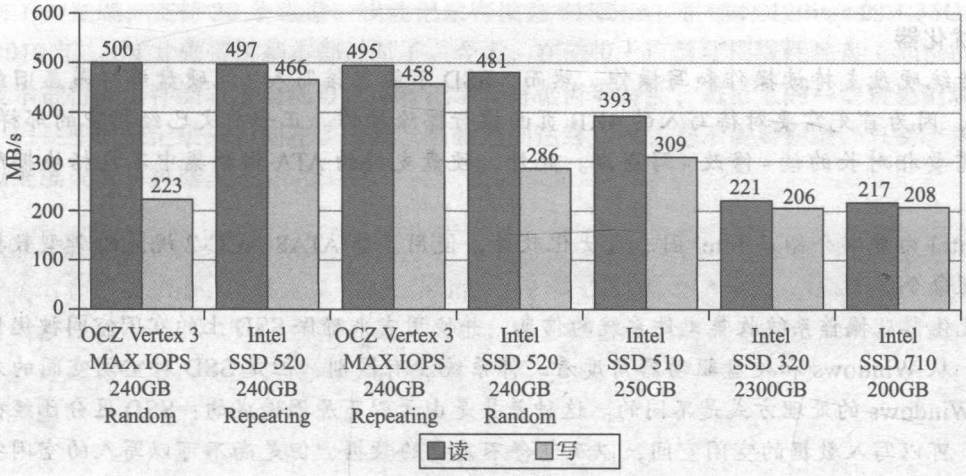
来自 storagereview.com 的图 3-38 给出了几种固态盘的顺序数据传输性能。这些数字表示对 2MB 数据进行顺序和随机的读取和写入操作。注意 SSD 几种不同模型之间的区别。还要注意写性能可能只有读性能的 50%。



a）2MB 数据块的顺序传输率（读 / 写）

图 3-38 SSD 随机读和写

⊖ Oliver B., "Comparison of 32GB SSD vs 10,000 RPM HDs", http://www.xlr8yourmac.com/.



b）2MB 数据块的随机传输率（读 / 写）

图 3-38 （续）

SSD 的第二个特征是写周期数量的限制。当闪存存储位元被擦除，环绕栅极的介电绝缘体被破坏，存储位元在一定数量的周期后将失效。此外，重复的写周期可以导致电荷捕获效应（charge trapping），从而导致存储位元停留在零状态。当闪存用于 BIOS、数码相机或 MP3 播放器的存储器时，闪存的耐久性不大可能成为问题，因为可忍受的典型最小擦除周期为 10000 ～ 100000 次的量级。然而，当闪存作为二级存储器时，效果就不一样了。

一种称为损耗均衡（wear leveling）[⊖]的技术通过将负载分散在整个设备中以克服闪存写周期限制产生的不利影响。它使用一种内存管理形式将驱动器控制器请求的块地址映射到存储器阵列中的物理块地址。控制器跟踪物理存储器中块被使用的频度，改变映射表确保所有物理块中的负载均衡。

Micron 的技术报告 TN-29-42 提供了损耗均衡的一个示例。假设某系统不使用损耗均衡技术，50 个数据块一小时更新 6 次。此外，假设这 50 个块可以使用存储器空间中的 200 个块。由于每一个块可以使用设备中 4 个块中的一个（即 200/50），因此其生存周期是 10000 周期 ×4=40000 周期。在更新速度为 6 个文件 /h 的情况下，存储器的生存周期大约是 280 天。然而，如果使用损耗均衡，存储器中所有 4096 块均可用，生存周期将扩大 4096/200 倍，约为 15 年。

有很多方法实现损耗均衡。例如，可以将存储空间划分为静态和动态区域。静态区域用于存储相对不太可能被改变的数据（例如，程序）。管理静态数据不需要损耗均衡。动态区域用于存储经常被改变的数据，并由损耗均衡软件来管理。

Toshiba 发布的信息表[⊜]得到了有意思的结论，128GB 的 Toshiba ML.C SSD 五年中估计的访问量可达到 80TB。该磁盘使用 NAND 闪存技术和多层次数据存储技术（每个位元存储多于一位的数据），与其他技术相比支持更少的写周期。如 Toshiba 所述，这相当于每天超过 20GB 的数据，这对于大多数移动电脑的终端用户应用程序来说都是巨大的数据量。如果 SSD 更大，可以改写的数据量也相应地增长。注意，基于 SLC 技术的 512GB SSD 每日可以支持 2TB 的数据访问量。

⊖ "Wear-leveling techniques in NANO flash devices", Micron Technical Note TN-29-42.

⊜ "Solid State Drives: Separate myths from facts," Toshiba America Electronic Components, Inc., V1.3, June 2009.

Intel 优化器

传统硬盘支持读操作和写操作。然而，SSD 不能直接写入（像硬盘那样覆盖旧的数据），因为首先需要对待写入的 4KB 页面执行擦除操作。正如前文已经指出的那样，SSD 需要相对长的读 - 修改 - 写周期。此外，硬盘支持的 ATA 指令集中不包括块擦除命令。

Intel 白皮书介绍了 Intel 固态盘优化软件，使用了非 ATA8-ACC-2 规范的新型数据集管理命令。

优化器从操作系统收集文件系统的信息，并使用它来确保 SSD 上的空闲空间被优化使用。从 Windows 和硬盘驱动器角度看文件系统没有区别，但是 SSD 对空闲空间的处理与 Windows 的处理方式是不同的。这种差异是由于以下原因造成的：SSD 区分已经被擦除，可以写入数据的空闲空间，以及包含不需要的数据、但是尚不可以写入的空闲空间。因为优化器通过擦除数据回收空闲空间并使其可以被使用，系统性能得到了改进。

注意，优化器并不从回收站中删除文件，这将使用户不能恢复意外删除的文件。回收站中的文件将一直保留，直到用户执行清空回收站（empty recycle bin）命令——此时优化器将回收文件占用的空间。

混合驱动器

到 2010 年，SSD 逐渐在高端应用场合中出现，但是它对大多数应用来说还是太昂贵。Seagate 试图通过混合驱动器来弥合快速但昂贵的硅器件与缓慢但廉价的磁盘之间的差距。其 2.5 英寸的 Momentus XT 将这两种技术结合在一起，提供具有 4GB 闪存（加上大容量的 32MB 的快速半导体缓存）的 500GB 磁盘。4GB 的闪存相对总容量来说较小，但板载软件监控磁盘的使用情况并缓存经常使用的数据。性能数据表明，混合驱动器相比传统的硬盘驱动器有较好的性能优势。其他测试已经表明，混合驱动器的启动时间（加载 Windows）相比传统的硬盘驱动器有 25% 的改善，且在关机时性能提高了 300%。

3.6 磁带

磁带提供了存储海量数据的手段。在数据每年增长约 60% 的情况下，档案存储就是至关重要的。在 20 世纪 60 年代和 70 年代的科幻电影中是不可能错过磁带驱动器的。机房中摆满磁带柜，一大盘磁带在其中旋转，然后停止不动，接着再次启动甚至反转。在那时候，首选的备份媒介为磁带，因为它相对便宜、可以存储大量的数据。

数据以多条平行磁道的形式存储在磁带上（通常为 9 条；分别记录 8 个数据位和 1 个奇偶校验位）。磁带记录技术几乎与磁盘技术相同，区别在于沿着柔软的磁带（通常为 2400 英尺长）上有一条 9 位宽的轨迹。磁带驱动器需要强大的引擎来快速旋转和停止磁带盘。直到 20 世纪 80 年代，磁带盘的直径为 10.5 英寸，典型的数据存储密度通常为 128 个字符 /in。当然，所有磁带系统的延迟都很长，因为定位所需数据需要将磁带在读 / 写头下方移动，这可能需要几秒甚至几分钟。因此，磁带完全是一种保存档案的介质。

磁带驱动器越来越小，磁带盒技术（类似音频磁带和 VCR 磁带）也在发展。为了在磁带上存储更多的数据，通过使用旋转的螺旋读 / 写头，信息沿着斜线存储在磁带上。同样的写机制被用于美国的 VCR。1972 年引入的 1/4 英寸磁带机（QIC）标准为开盘式录制设备

带来了大发展，支持30条磁道、线性记录密度为51Kb/in、带速为120in/s的1.35GB磁带。到2010年，1/4寸磁带机基本都过时了。今天，在磁带上广泛使用线性蛇形（serpentine）记录技术进行数据存储。术语蛇形意味着记录具有锯齿形特性，磁带上的一些轨迹记录从左到右，另一些轨迹记录从右到左。因此，当磁道到达终点时，它不需要回退，只要改变方向就可以继续读或写数据。

磁带库技术

磁带库（ultrium）技术是由Quantum、HP和IBM联合开发的一系列线性磁带开放（Linear tape-open，LTO）标准的通用名称，它为那些被迫购买专有记录技术并被迫选择某个供应商的用户提供了共同的标准。

LTO标准具有3个重要特点：

1. 它们会定期更新，以与需求和技术发展保持同步。
2. 它们使用线性蛇形技术（而不是螺旋记录）。
3. 它们提供了当今所需的很高的存储容量。

关于磁带和盒式磁带技术已经过时是当今普遍的看法。认为磁带已经消亡是更为夸张的观点。2011年1月，HP公司报道，其LTO磁带驱动器的全球市场在2008年年末至2009年年末下降了约30%，但到2010年年底（相对2008年的数据）增加了45%。这相当于磁带使用的复兴。此外，磁带具有比磁盘驱动器更低的总体拥有成本（total cost of ownership，TCO）。大型组织，如执法或医疗机构，有大量数据需要存储。总体拥有成本包括设备、介质、占地面积、维修以及能耗。针对大型组织12年期间的数据，基于磁盘存储的TCO可能是基于磁带存储的TCO的15倍[⊖]。

直到20世纪90年代末，LTO（Linear tape-open）标准才被提出。open这个词表明该标准是开放的，不像早些时候由IBM和HP提出的专有标准。LTO的第一个标准——LTO-1，导致了100GB的盒式磁带的出现。到2010年，已经发布了LTO-6标准，它支持1.5TB的容量，使用896个磁道，线性密度为15142b/mm，数据传输率为140MB/s。LTO标准已经扩展到LTO-8版本，支持12.8TB的盒式磁带，该标准尚未公布。LTO-5标准的1.5TB盒式磁带并不便宜。2011年，盒式磁带费用是50美元的量级，这与硬盘存储器类似。

制造现代高密度磁带并不容易。轨道之间的距离很小，使得*磁带方向稳定性*（tape dimensional stability）成为一个问题。磁带在温度和湿度的影响下可以改变尺寸。这意味着，在写数据时与并行磁头排列一致的磁道与稍后读取磁带时的磁头排列可能不一致。这导致需要高度稳定的带基板。例如，一些制造商使用聚酰胺，它是凯夫拉芳纶纤维材料的近亲。该材料的一个额外的好处是，如果遭受攻击，可以用它织成防弹背心。

LTO盒式磁带的一个有趣选项是*一次读多次写*（write once read many，WORM）模式，允许盒式磁带只能写入一次。底层的记录介质与盒式磁带相同，但物理接口可以防止重写，它具有一个不同的伺服磁道来校验数据没有被修改过。这些盒式磁带存储的数据需要防止篡改是因为其法律用途而不允许修改。

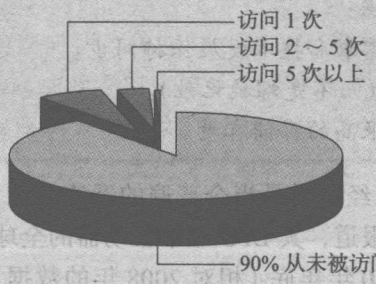
对于数据存储来说，磁带仍然是有吸引力的存档机制，因为它可拆卸也可移动。下一节将介绍的光学存储介质也是可拆卸和可移动的，但尚未提供多个TB的数据存储能力。直

⊖ Clipper Notes, report #TCG2010054LL, December 2010.

到20世纪90年代初，磁带都使用氧化介质和铁氧体读/写头。在过去的20年里，磁头技术的进步（与硬盘驱动器磁头的进步相同），大大改善的磁性介质，磁道密度的增加，使得磁带系统的面密度增加了超过3个数量级。例如，Sony公司开发的磁粒子技术通过将30nm的金属粒子封装在合金/陶瓷涂层中以防止氧化和渐进的数据丢失来提供高密度记录[⊖]。同样，为了减少退磁效应需要非常薄的磁层。当前磁层约为200nm的量级（人类头发的直径为60000nm）。

悖论，悖论，最巧妙的悖论

下面取自Ultrium报告的幻灯片很有趣，因为它得出结论：90%的网络数据从来不会被访问。如果这些数据存储在磁盘中，其存储消耗和能量消耗都是巨大的。



来源：Bruce Master, "The Evolving Roles of Tape and Disk," *Ultrium consortium presentation*, 2009.

3.7 光学存储技术

利用物质磁性来存储数字信息一点也不奇怪，这是因为磁性固有的二值属性，它已经在相当长的一段时间里被应用于各种系统中。基于光来实现数字存储记录机制的应用早已是见怪不怪。照相底片已用于存储模拟图像许多年了。1834年，Henry Fox Talbot使用浸润氯化银后的相纸制成永久的负面图像（底片）。正面图像是将负面图像通过接触印刷（contact printing）到另外一张感光纸上。

传统化学照相技术确实可以用来存储数字数据。假设有一张2in×2in覆盖着感光材料的塑料。可以使用与激光打印机相同的技术在其上存储（例如2×1200×2×1200个点）和使用数据，并使用文档扫描仪技术来读取数据。这种技术可提供约720000字节/页的存储容量，甚至可以将速度提高10倍达到7.2MB/页的容量。

光学数字存储介质曾经是蓄势待发而且是不可避免的一项发明。问题只是看哪种技术将占主导地位，哪些公司将抢先占领市场。本章将介绍3种数字计算机所使用的光记录机制：CD、DVD和蓝光光盘。因为使用相同的底层技术，这里将依次介绍它们。它们之间的差异主要就是规模；随着时间的发展，可以将这些特性按比例缩小至存储数据的盘片上。

3.7.1 数字音频

光存储的出现比许多人认为的要早许多。Philips在20世纪70年代早期就开发并在80年代推出了激光影碟。激光影碟具有与现今CD（compact disk）相同的特性，只是激光影碟

⊖ "Sony, metal particle, and A3MP tape: Nanoscale technology for terabyte storage," Sony Electronics Inc., Park Ridge NJ., January 2009.

使用模拟技术。激光影碟系统是只读的，很少有电影存储在其上。激光影碟的费效比不高且不够成熟，因此是一种失败的技术。

人们所熟知的数字音频很大程度上要归功于两大消费电子行业巨人之间的合作。20 世纪 70 年代，Philips 和 Sony 联手开发了光存储技术，特别受益于 Philips 在光学技术上的丰富经验和 Sony 在编码和纠错机制方面的专业知识。

Philips 和 Sony 的一个重要成就是基本记录和回放机制的标准化。因为技术和各种参数的复杂性，这种标准化绝非易事。两个基本参数是 CD 尺寸和采样频率（sampling frequency）。盘片的直径是 12cm，采样频率选择为 44.1kHz。虽然这似乎是一个奇怪的数据，但这是高质量音频所需的采样频率，电视系统也使用相同频率。据报道，CD 的基本参数必须兼顾 74min 的音乐时长，这足以在单张 CD 上容纳冯·卡拉扬版的贝多芬第九交响曲。另一方面，参与 CD 研发的一些人否认了这个故事的真实性。不管怎样，希望它是真的。

第一个基于光的实用、低成本、高密度光存储机制是 CD，它在 1981 年推出，用来以数字的方式存储高质量的声音。在引入 CD 之前，音乐存储在磁带或黑色乙烯密纹唱片（Long Playing record，LP）上。LP 又大又重，每面只能保存 25min 的音乐。此外，它是模拟介质并非常容易损坏（例如，划痕和扭曲）。

CD 的结构类似于磁盘，它沿着某条轨迹存储信息。与硬盘不同，CD 的轨迹是连续的螺旋轨迹，如图 3-39 所示。螺旋轨迹约 20000 圈，约相当于 3 英里。有效的道密度为 16000 圈 /in，理论上的最大面密度是 1Mb/mm^2 或 80Mb/in^2。

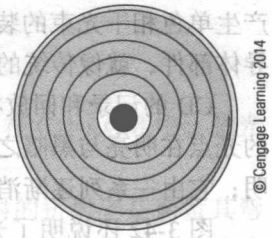


图 3-39 螺旋轨道

CD 上的数字信息印制在 1.2mm 的塑料盘片表面，由称为凹坑（pit）的缺口的组合形式记录成不同的长度。凹坑具有反光层和保护层。表面没有缺口的地方被称为平面（land）。盘片中央孔的直径为 15mm，数据记录在离中心 25 ～ 58mm 的盘片上。

读取数据的过程是首先在盘面上照射形成一个微小的点，然后检测从表面反射回的光线强度。每当激光点从凹坑移动到平面上时反射光强就会发生变化，反之亦然。

图 3-40 说明了 CD 表面的结构。创造凹坑这个词在词法上受到挑战。凹坑从盘片表面长出，民间说法叫作肿块（bump）。凹坑一词指的是从盘片上部的标签层向数据记录层看去是一种缺口。但读取数据的激光束看到的是*肿块*而不是凹坑。图 3-40 表明，光盘有 4 层；光盘本身厚约 1.2mm，由透明的聚碳酸酯塑料构成。凹坑被涂上一层薄的铝，后面跟了一层保护性的丙烯酸层，标签印在最上面。标签这面对划痕和磨损比光滑的那面更敏感。

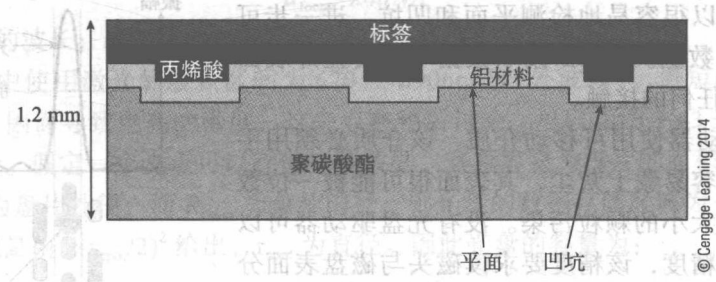


图 3-40 CD 的结构

透明 CD 表面的划痕不会破坏数据，尽管由划痕导致的光散射可能增加错误率。由于数

据是沿着轨道存储的，因此介质对径向划痕不太敏感而对圆周划痕敏感。如果要清洁光盘，擦拭应该沿着径向！

CD 的面密度是轨道一圈记录的数据位数与轨道密度的乘积。每一位的物理尺寸取决于照射到光盘表面光点的大小。图 3-41 展示了 CD 表面凹坑和平面的结构和尺寸。肿块约为 0.5×10^{-6}m，轨道间隙为 1.6×10^{-6}m。肿块（凹坑）的高度是 1.25×10^{-7}m。

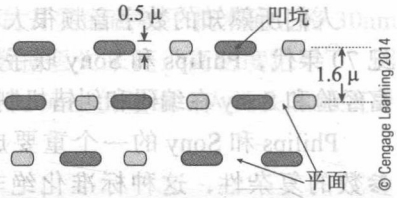


图 3-41 CD 的表面

3.7.2 从 CD 中读取数据

前文已经说过，将一束激光照在凹坑和平面构成的轨道上，通过反射光强来读取数据。通过应用现有的半导体技术可以将凹坑做得非常小。不幸的是，创建真正微小的点是很难的，因为大多数光源产生的光都会发散。

为了创建和控制尽可能小的光点，必须使用相干（coherent）光和单色（monochromatic）光光束。如果光波都具有相同的频率则为单色光源，不像白光那样包含整个可见光谱的各种频率。如果光波像具有相同频率那样具有相同的相位（phase），则光波为相干光。激光器是产生单色相干光束的装置，因此，可以使用激光照射 CD 的表面。激光器是一种低成本的半导体部件，就像传统的 LED（发光二极管）那样。

CD 的记录和回放机制十分巧妙。图 3-42 展示了在盘面上激光点的能量分布。一个完美的光点在明亮与黑暗之间有明显的边界。然而，由于光的波动性质，光点的边缘并非界限分明；它由一系列逐渐消退的光环组成。

图 3-42 还说明了光点、轨道、凹坑、平面的相对尺寸。当光照射 CD 表面平坦的区域（平面），光线的大部分被反射回来。假设光点的大小比单个肿块要大。当光照射到表面的肿块时，一部分光从肿块表面反射，另一部分光从肿块附近的表面反射。

如果光源不是相干光，光从表面（即平面）反射还是从肿块（即凹坑）反射的区别不大。然而，CD 中，肿块相比平面的高度是激光波长的四分之一。假设从光源发出至盘片平面然后反射回观测器的总距离为 x。照射到肿块底部的光线的传输距离稍长一些，即每个方向都会长 $\lambda/4$。因此，从肿块底部返回光的总的传输距离为 $x+\lambda/2$。由于光点没有完全覆盖一个凹坑，从凹坑反射回的光与从平面反射回的光的传输距离相差为光波长的一半，此时光波会消失。这是相干光的一种性质。

从平面返回的光线很强，但从凹坑返回的光线衰减严重。因此，可以很容易地检测平面和凹坑，进一步可以从盘片中读取数据。从 CD 读取数据不需要盘片表面与读写头部件有任何的接触。

因为 CD 驱动器使用可移动介质，该介质必须用手指触碰，存放时容易蒙上灰尘，其表面很可能被一位数据对应存储尺寸大小的颗粒污染。没有光盘驱动器可以实现磁盘的工程精度，该精度要求读磁头与磁盘表面分开μm 量级的距离。幸运的是，光点尺寸的问题可以利用制造 CD-ROM 的洁净碳酸酯的光学性质来解决。

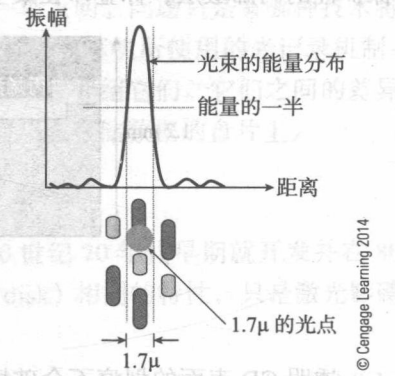


图 3-42 光束的能量分布

图 3-43 展示了 CD 表面污染的影响是如何被显著降

低的。半导体激光器发出的光经过物镜聚焦到数据盘片的表面。照在凹坑和平面上的实际光点大小为 1.7μm，而照在盘片上层透明表面的光点大小为 800μm。透明表面光点的大小为照在凹坑和平面上光点的近 500 倍。这意味着表面轻微的瑕疵对聚焦的影响很小，所以光盘表面对污染相对宽容。

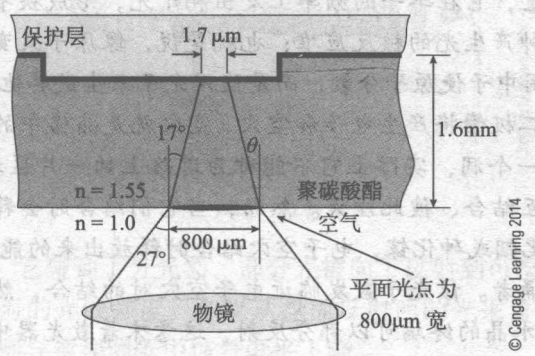


图 3-43　CD 光学和光点大小

照在凹坑和平面上光点的大小可用下式计算：

$$r_{spot} = 1.22\frac{\lambda}{NA}$$

其中：NA 为物镜的数值孔径（numerical aperture）；λ 为激光的波长。在 CD-ROM 中，激光的波长为 780nm，数值孔径大约是 0.45。

数值孔径通过 $NA=n\sin\theta$ 计算，其中 n 是聚碳酸酯的折射率。材料的折射率是衡量其弯曲光线的能力，定义为真空中的光速除以介质中的光速。数值孔径用来衡量透镜系统的聚光能力，决定了其分辨能力和景深。也可以用相机来解释，其镜头的数值孔径与光圈（f-stop）值成反比。

先做一个近似计算，图 3-43 的几何计算表明盘片厚度和表面光点尺寸的关系式 $\tan\theta=s/2d$，其中 d 是盘片的厚度，s 是光点的直径。对于较小的 θ 值，$\tan\theta=\sin\theta$，因此，

$$s=2d\sin\theta=2dNA/n$$

其中，d 为盘片的厚度；NA 为物镜的数值孔径；N 为盘片材料的折射系数。

设计 CD 读取部件需要考虑的另一个重要因素是景深（depth of focus）；也就是说，光点可以聚焦的范围。景深越小，激光点就越难跟踪移动着的盘面。景深可以表示为：

$$d_{focus}=\lambda\cdot n/NA^2$$

其中，λ 为激光的波长；n 为材料的折射率；NA 为数值孔径。

CD 驱动器中使用激光的波长区间为 670 ～ 690nm（红外波段）。更短的波长允许更小的凹坑和平面，因而导致更高的密度，这一点在稍后讨论。可以用图 3-43 来估计 CD-ROM 的最大理论容量。假定一个光点可以存储一位数据，其总容量为盘片的面积除以该光点的面积。如果 R_{outer} 为盘片外径，而 R_{inner} 为盘片内径，则光盘的数据存储区域为 $\pi R_{outer}^2-\pi R_{inner}^2$。

光点的面积是由 $\pi(r_{spot}/2)^2$ 给出，r_{spot} 为直径。因此光盘的容量为：

$$容量=\frac{R^2_{outer}-R^2_{inner}}{\left(\frac{1.22\lambda}{2NA}\right)^2}$$

如果将典型值代入此公式[⊖]，可以得到CD的理论最大容量为2.5GB。实际上，不可能实现该容量，部分是因为光盘需要存储纠错码。

激光器

激光器是一种装置，它在单一的频率上发出相干光，形成狭窄、不发散的光束。本质上说，激光器是一种产生光的核反应堆；也就是说，像原子裂变，它依赖于一种连锁反应。不过它不是使用中子使原子分裂，而是使用光子来生成其他光子。

电流穿过半导体二极管将产生电子和空穴（空穴就是晶格中的一个区域，其中没有电子，就像道路上的一个洞，实际上它不能称为道路上的一片区域）。当一个电子和空穴相遇时，它们便相互结合、彼此湮灭。然而，当它们结合时会释放能量。通过使用合适的半导体材料如磷化铟或砷化镓，电子空穴结合时释放出来的能量就是光子。

这就是那个始作俑者。该光子激发临近电子空穴对的结合，然后产生相同相位和频率的光子（光放大）。水晶的终端可以部分反射，这意味着激光器中的光会发生端到端的反射，从而产生更多的光子。

激光器（laser）是辐射的受激发射光放大器（light amplification by stimulated emission of radiation）的缩写，它指所有通过连锁反应释放更多光子的过程/机制。考虑存储设备中激光器的输出功率，在只读设备中通常为5～10mW，而在刻录设备中可达250mW。外科手术中使用的激光器功率为100W，而在军事上激光武器的功率为100kW。

1. 光盘的速度

音频CD的速度由从盘中读取所需数据的速度来衡量。与150KB/s的数据传输率相对应的速度被称为1X（一倍速）。以这个速度，读头下盘片表面运动的速度为1.25m/s。因为用户需要尽可能快地读取数据，光驱从产生后便不断变得更快。4X驱动器可以以标准音频光盘4倍的速度提供数据。工作在48X的驱动器如今已经司空见惯。然而，对驱动器进行基准测试的组织公布的测试结果表明，这些驱动器不能提供用户期待的持续数据传输率。48X光盘驱动器的速度并不是1X光盘驱动器的48倍。

与硬盘不同，第一代的CD驱动器以恒定线速度（constant linear velocity，CLV）运转。恒定线速度意味着读头下方盘片表面的速度是常量。硬盘以恒定角速度（constant angular velocity，CAV）运转。因为盘片中心和边缘的半径有很大差异，读取光盘时，以恒定速度读取数据所需的旋转速度各不相同。该特性限制了CD-ROM驱动器的速度。

现代CD-ROM驱动器在很大程度上放弃了纯粹的CLV，因为在高速旋转时很难实现。一些驱动器使用了双模式，在靠近光盘中心时使用恒定角速度读取数据，而在靠近光盘边缘时使用恒定线速度来读取数据。然而，当前的光驱为DVD（或蓝光）驱动器，它可以向后兼容，可以读取CD。

2. 光学读头

下面介绍从光盘中读取数据的完整光学系统。图3-44显示了激光从激光器发出到达盘片表面，然后再回到光电传感器的路径。本质上说，从激光器发出的光穿过了一系列镜头，

⊖ Lane, P. M. and Van Dommelen, R., "Compact Disc Players in the laboratory," IEEE Transactions on Education, Vol. 44, No. 3, Frbruray 2001, pp47-60.

聚焦到盘片上的一个点。光按照原路从光点反射回来，反射光的强度取决于光点照在凹坑还是平面上。

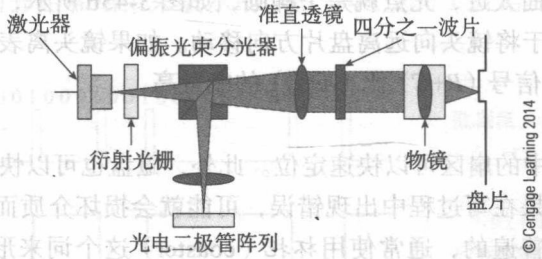


图 3-44 读数据

当光从盘片返回时，它碰到了分光器，部分光被向下反射到可检测光的传感器。光电二极管可以检测从盘面反射的光线强度。如果从盘面反射回的光来自肿块，部分光多传输了 $\lambda/2$ 并变成 180° 异相，则导致探测器中信号电平下降。

3. 聚焦和跟踪

为了可靠地读取数据，光驱的读头必须精确地对准螺旋光道。沿着半径方向可以很容易地移动光学装置。但是很难将激光以必要的精度对准所需的位置。该问题对在 XY 平面上寻道以及在 Z 方向聚焦时都存在。

读头中的物镜安装了平衡环，可以在两个平面运动：左右运动来实现跟踪；内外运动来实现聚焦。使用电磁铁的磁场可以定位物镜来进行高精度的跟踪和聚焦。

数据跟踪的方法很巧妙（有几种变形）。为了理解跟踪和聚焦，必须了解光传感器。图 3-45 显示了全部 6 个光学传感器。来自激光器的光首先通过衍射光栅（diffraction grating），它是一种具有平行刻线的透明材料。衍射光栅的作用是把单束光分割成一个主光束和两个侧光束。

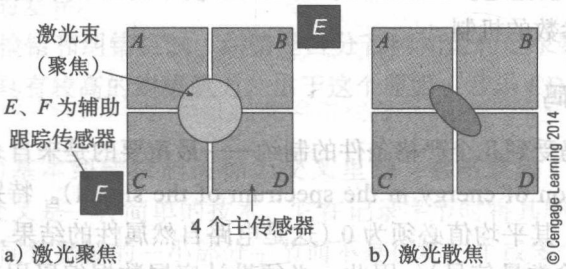


图 3-45 使用光传感器读数据以及进行聚焦 / 跟踪

传感器 A、B、C 和 D 接收主光束。传感器 E 和 F 接收两个侧光束。这两个侧光束传感器的输出相减可以获得跟踪误差（$track_{error}=E-F$）。如果光束集中，跟踪误差为 0。如果 $track_{error}>0$，则光束需要向左移，如果 $track_{error}<0$，则光束需要向右移。

主光束落在传感器 A、B、C 和 D 上，这些传感器输出的总和用来重新生成来自盘片的数据。一对传感器之间的输出差异可以用来对光点进行聚焦。

在图 3-45 中，来自衍射光栅的光束通过准直透镜变成了平行光束，再通过四分之一波片以及一个物镜将光束聚焦到盘片上。这些光束沿着同样的路径返回，然后通过分光器到达 6 个传感器。

物镜是圆柱形（cylindrical）而非球面（spherical），这导致了散光；也就是说，焦点在垂直面和水平面上是不同的。如果光束聚焦，光点就是圆形，所有4个中央传感器将获得相同的能量。如果镜头离表面太近，光点就是个椭圆，如图3-45b所示，信号（$A+D$）比（$B+C$）的能量高。该差值可用于将镜头向远离盘片方向移动。如果镜头离表面太远，散光的效果则是旋转90°的椭圆，使信号（$B+C$）比（$A+D$）的能量高。

4. 缓冲区欠载

与光盘相比，磁盘中的扇区可以快速定位。此外，磁盘也可以快速地写入。将数据写入光学介质更为复杂。如果在写过程中出现错误，可能就会损坏介质而被丢弃。20世纪90年代末，损坏CD是非常普遍的，通常使用杯托（coaster）这个词来形容使用损坏的CD-R[⊖]作为咖啡杯的垫子。当光驱写CD-R时需要连续的数据流。数据流中最微小的空隙也是致命的。因为在写CD时，来自主机处理器的数据被存储在CD驱动器的缓冲区中，出现缓冲区空的情况被称为缓冲区欠载（buffer underrun）。

这种缓冲区欠载问题是记录CD最大的问题之一，每当CD驱动器较快而数据收集较慢或处理器在写CD时意外中断就会发生。到2000年，CD-R和CD-RW驱动器制造商通过提供缓冲区欠载解决方案而相互竞争。

首先解决该问题的造商之一是Sanyo，它用BURN-Proof™（防止缓冲区欠载）技术这个术语来描述其系统。Sanyo的机制是定期检查数据缓冲区的状态。如果缓冲区清空得过快，CD-RW的固件将干预处理该情况。用户所要做的就是写完当前帧，然后等到缓冲区再次被充满。Plextor扩展了Sanyo的BURN-Proof技术，它定期抽样记录的数据来检查其质量，允许驱动器将盘片的速度更改为更为合适的值。

Yamaha公司开发出一种缓冲问题的解决方案，称为SafeBurn™。Yamaha的系统既避免出现缓冲区溢出也进行写速度的优化。SafeBurn技术使用相对较大的8MB的缓冲（当时其竞争对手只使用2MB的缓冲）。如果缓冲区中的数据量开始降到危险的水平，写操作像BURN-Proof技术那样被挂起。Yamaha的SafeBurn技术的第三个组成部分是一种检查被写盘片特性以优化记录参数的机制。

3.7.3 底层数据编码

盘片上数据的编码受到几个严格条件的制约——最重要的是来自光学传感器的信号光谱的能量分布（distribution of energy in the spectrum of the signal）。特别是，信号中应该没有直流分量；也就是说，其平均值必须为0（这是电路自然属性的结果，它只能传输随时间变化的信号，而不能传输常量信号）。因此，必须设计底层数据编码用来控制连续的0位或1位的数量并允许从数据信号中提取时钟信号（自同步能力）。

源数据以8位一个字节为单位进行存储。这些数据字节都通过8～14位调制（EFM）被编码成14位。每个14位的编码可以表示2^{14}=16376个不同的值，尽管其中只有2^8=256个值被使用。EFM是一种游程长度受限（run length limited）码，它减少了信号的带宽。EFM编码确保盘片上的数据流中有少于10个且多于2个连续的0。事实上，有267个合法（legal）的14位编码符合规则：数据流中连续的0的数量介于3和9之间（包含3和9）。这意味着调制机制有267-256=19种编码是合法的、但没有用来描述有效的数据字节。因此这

⊖ CD写后就不能更改。CD-R是可以写一次的CD。CD-RW是可以写、擦除，然后再重写的CD。

些代码可以在数据流中用作特殊标记。

数据中逻辑 1 的值被解释为来自盘片信号状态的变化（即从平面变为凹坑，反之亦然），1 就是通过记录凹坑的开始和结束来表示的。图 3-46 说明了数据流和盘面记录情况之间的关系。

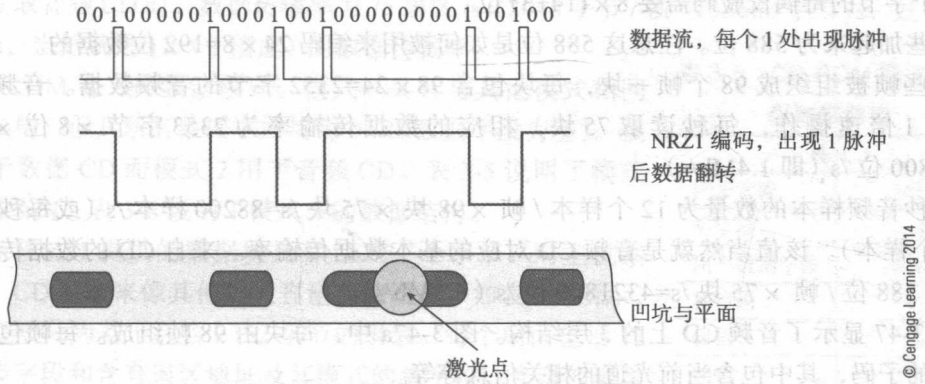


图 3-46　数据、编码和轨道结构

不幸的是，很可能会出现一组的结束和下一组的开始被解释为有效编码字的情况。为阻止这种情况的发生，在每 14 位一组间需要增加 3 位分隔符。这 3 位被称为合并位（merging bit）。组成合并码的 3 位并不总是相同的。选择特定的合并码通常用来确保 1 和 0 的数量基本相同（该限制确保从光头中读出的数据中没有直流分量）。

CD 上的底层数据结构十分复杂，这是由于纠错编码机制导致的，它将光盘记录从本质上不可靠的存储转换为可靠的存储。光介质暴露在空气中并通过用户转递使得它们很容易受到污染。而且，记录介质也有制造方面的缺陷，回放机制也受到振动和其他干扰的影响。进而，相对于硬盘而言，从光盘中读取的数据很可能会有较高的错误率。确实，光盘与磁盘出错率有好几个数量级的差异。

光介质使用集中检错和纠错机制。纠错是区分音频光盘和记录数字数据的 CD-ROM 的主要特点。数据光盘具有较高的纠错能力。出于这个原因，音乐 CD 可以存储比数据光盘更多的数据位。

光存储中使用的基本纠错机制被称为交叉里德－索洛蒙码（Cross Interleaved Reed-Solomon，CIRC）。交叉是一种简单的技术，它在记录字节时将其顺序重新排列，以确保长的突发错误只影响不同数据帧的一小部分字节而不是损坏一个或多个相邻数据帧的许多位。换句话说，长的突发错误被重新分配为多个短的突发错误。这些短的突发错误通常可以被纠正。

下面来看音频光盘的格式。数字化声音信息最开始为多个 16 位数值对，每个样本是分别对左和右声道采样的结果。这些数据被组合为 24 字节的帧。CIRC 机制使用这 24 字节的数据帧产生 28 字节的输出。额外的 4 个字节是 RS（里德－索洛蒙）编码器创建的纠错信息。28 字节的数据然后被传递给第二个 RS 编码器生成 32 字节的帧，进一步添加了 4 个字节的纠错信息。这种编码机制旨在防止一位和多位突发错误。在帧中添加了一个字节的子码，其原因稍后解释。

使用 8 ～ 14 位编码并在前面增加 24 位同步码来对信息进行编码。在每个数据字节间增

加3位的合并码，但同步码内不需要。因此，总的位数为：

24个同步位+3个合并位

1个字节的子码需要1×(14+3)位

24个字节的数据需要24×(14+3)位

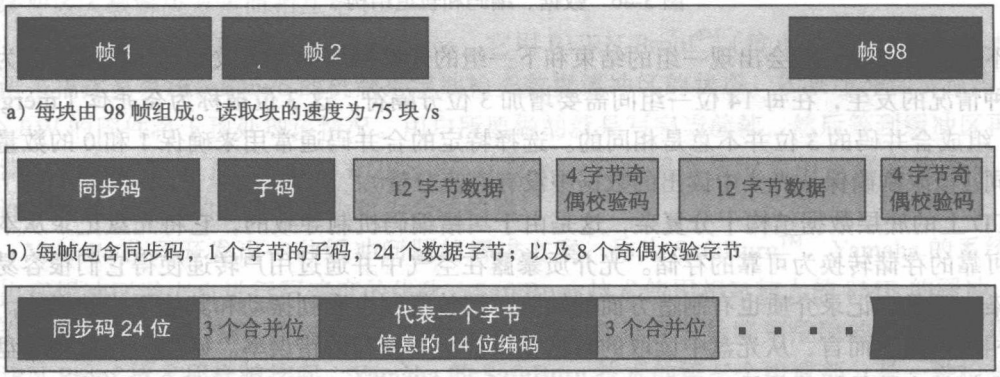
8个字节的奇偶校验码需要8×(14+3)位

这些加起来为588位。注意这588位是如何被用来编码24×8=192位数据的。

这些帧被组织成98个帧一块，每块包含98×24=2352字节的音频数据。音频CD驱动器以1倍速操作，每秒读取75块，相应的数据传输率为2353字节×8位×75块/s=1411800位/s（即1.4Mb/s）。

每秒音频样本的数量为12个样本/帧×98块×75块/s=88200样本/s（或每秒每声道44.1K个样本）。该值当然就是音频CD对应的基本数据传输率。来自CD的数据传输率为98帧×588位/帧×75块/s=4321800位/s（4.3Mb/s）。

图3-47显示了音频CD上的3层结构。图3-47a中，每块由98帧组成。每帧包含一个单字节的子码，其中包含当前光道的相关信息等等。



a）每块由98帧组成。读取块的速度为75块/s

b）每帧包含同步码，1个字节的子码，24个数据字节；以及8个奇偶校验字节

c）帧的精确结构：8位数值被编码为14位，每组间包含3个合并位

图3-47 音频CD上的数据结构

需要98帧来构成完整的子码。子码的位代表了8个通道P、Q、R、S、T、U、V和W。P子码用于指定音轨之间的分隔符，Q子码包含盘片的目录以及音轨的位置。其他子码一般不使用。子码由一块中的98个子码字节组成，开始有两个特殊的14位同步码，与正常的256个数据代码不符。这意味着子码块可以被唯一标识。子码块也包含自己的16位循环冗余码来进行错误保护。

图3-47b显示了由24字节音频数据、同步头、子码字节和由里德－索洛蒙编码器生成的奇偶校验字节组成的帧结构。图3-47实际上是误导，因为数据的顺序已被CIRC编码器打乱。

图3-47c说明了帧的位结构，包括24位唯一的同步码和连续的14位信息单元，每个单元被3位合并码隔开。

CD-ROM的编码更加复杂，因为加入了额外数据纠错机制来进一步降低不可纠正的错误率。音频信号比数字数据更能容忍错误。在音频样本流中偶尔丢弃一位不会产生大问题，这是因为由此产生的声音干扰会很短暂，可能部分被音频处理电路过滤掉。在数据信号中丢弃即使是一位也会产生严重的影响，特别是数据表示代码的时候。

CD-ROM 的结构类似于音频 ROM，其目的是提供两种系统之间的兼容性。因此，使用相同的帧结构和底层编码。98 帧的一块被称为扇区（sector），可以存储 2048 字节的用户数据。这种安排意味着，2336 个音频字节可以存储在一个扇区中，2336-2048=288 个字节可用于额外的纠错。

当读取音频 CD 时，数据传输率为 75 扇区 /s × 2048 字节 / 帧 =153600 字节 /s，这大约为 150KB/s，当然就是 1X（1 倍速）的数据传输率。

CD-ROM 有 3 种数据模式。模式 0 具有与其他模式相同的基本结构，但其所有数据字段均为 0（即数据字段为空）。模式 1 用于数据 CD 而模式 2 用于音频 CD。表 3-5 说明了模式 1 的数据结构，表 3-6 说明了模式 2 的数据结构。

由于模式 1 提供的数据块具有 $2K=2^{10}$ 个字节，对主机系统来说，CD 看起来像其他的大容量存储设备。如表 3-5 所示，数据轨道被分为若干包含 2352 字节的扇区。每个扇区包括 12 位的同步字段和含有扇区地址及其模式的头字段。光盘的扇区地址不同寻常，因为它通过时间（time）来定义。这类似于通过飞机从起点以恒定的速度到达指定地点所花费的时间来计算飞机的位置。地址表示为分：秒：扇区。分字段要超过 $A0_{16}$，这样 3 字节编码 $C0_{16}$：32_{16}：21_{16} 将代表位置 32 分 50 秒 33 扇区；也就是说，分字段的值比实际值大 $A0_{16}$。在模式 2 的帧中，所有 2336 个字节都可用于用户数据。这种模式可以用于视频或音频编码。

表 3-5 CD-ROM 模式 1 的扇区数据结构

模式 1	
同步字段	12 字节
头字段	4 字节
用户数据字段	2048 字节
检错码	4 字节
保留	8 字节
纠错	276 字节

表 3-6 CD-ROM 模式 2 的扇区数据结构

模式 2	
同步字段	12 字节
头字段	4 字节
用户数据字段	2048 字节
附加数据字段	288 字节

3.7.4 可记录光盘

可记录光盘是“橙皮书”[⊖]第 II 部分的主题，第一款 CD-R 机器于 1993 年首次出现。可记录 CD 驱动器与只读变种的区别不是很大。主要的区别是介质。可记录 CD 使用双层技术，在染色层后有反射层。如果光照射到染色层，它就会被吸收；如果它照到反射层，它就会被反射回来。记录数据的方法是使用激光在染色层烧一个洞至反射层。由于这个原因，人们有时使用烧录 CD 来表示向空白 CD 写入数据。

最初的染色剂是基于青色素的。今天，人们使用更现代的染色剂来适应由于紫外线而导致的数据退化和更高速的写操作。反射层可能是黄金，这是由于其良好的稳定性和抗腐蚀能力。CD-R 光盘在制造时印制的螺旋轨迹具有与 CD 相同的宽度和间距。

1. 可重写 CD

为使 CD-RW 技术与现有的 CD 驱动器兼容，有必要找到一种方法，沿着光轨创建和删除不同反射率的区域。两个主要的方法是相变（phase-change）和磁光（magneto-optical）技术。Panasonic 和其他公司开创了磁光存储。然而，CD-RW 设备已普遍采用相变技术。

增加聚焦激光束的功率可以在本地加热光盘表面的数据记录层。该层包含银、铟、锑和碲的混合物，它存在两个稳定状态：晶态（crystalline）和非晶态（amorphous）。当这种材料处于晶态时，它反射激光的能力比非晶态时要强。

⊖ 橙皮书发表于 1990 年，是 Philips 和 Sony 定义的一系列可记录 CD 标准的非正式名称。在红皮书（CD 音频）、黄皮书（CD-ROM）和蓝皮书（增强 CD）中还有其他光存储器标准。

CD-RW 光盘本身与传统的 CD 类似。衬底是 1.2mm 的聚碳酸酯盘，光轨（即螺旋槽）塑造在盘片上并具有时间信息。记录层夹在两个介电层（dielectric layer）之间，可以在相变层写或擦除过程中被加热时控制其热特性。数据层和介电层之后是反射层。

CD 标准

随着 CD 的发展，不断出现新标准以适应其特性和应用的变化。传统上，每组新标准被称为书（book），特定的书被赋予一种颜色。第一个标准是为紧密光盘数字音频系统制定的红皮书标准。黄皮书标准针对用来存储数字数据的 CD-ROM，黄皮书标准进行了扩展，包括定义了包含计算机数据和编码的音频和视频信息的 CD-ROM 格式的 CD-ROM/XA 扩展标准。

下一个标准是绿皮书，它定义了交互式紧密光盘（Compact Disc Interactive，CD-I）的格式。CD-I 为广大消费者的交互多媒体应用程序提供了平台。

橙皮书提供了一次写紧密光盘的标准。欧洲计算机制造商协会创建的 ECMA 168 标准包含了橙皮书。该标准统治了一次写 CD 和 CD-ROM 市场，包括多区段记录，扩展文件名约定以覆盖 Unix 风格的文件名。最后，蓝皮书包括了激光唱盘的标准，用于存储视频和声音数据，但其现在已基本过时。

橙皮书 CD-R 标准将 CD 分为系统使用区（System Use Area，SUA）和信息区，SUA 定义了光盘上数据的格式和类型。SUA 又进一步被分为功率校准区（Power Calibration Area，PCA）和节目存储区（Program Memory Area，PMA）。PCA 为激光提供了试验场。该区域允许驱动器为每个记录光盘校准激光功率。记录光盘被校准后将某位设置为 1。

保存用户数据的信息区从引入包含 Q 子通道目录开始。在引入区进行同步。在数据后就是引出区，它们包含了数据并定义 CD 的末尾。

高山（high sierra）文件标准在主机操作系统和 CD-ROM 之间提供了一个接口。该标准被国际标准组织接受为 ISO 9660 标准，适用于只读介质。它省略了随机访问读/写介质所需的机制。

高山文件系统是层次化的，每个 CD 都具有根目录。因为 CD 本质上是一种慢速的介质，层次结构可能会使在目录间搜索花费很长一段时间，高山文件系统在每卷中使用单独的路径表来描述文件层次结构并加以缓存。文件命名约定比 Windows 更严格，文件名限定为 8 个字符、一个点以及 3 个字符的扩展名。此外，文件名不能包含除了下划线之外的特殊字符。

ISO 9660 标准为 CD-ROM 设计，但对可记录与可改写介质来说并不理想。ISO 13346 新标准，设计用来支持可记录的机制。该标准也被称为通用盘格式（Universal Disk Format，UDF）。

ISO 13346 最重要的特性是它支持包写入（packet writing）。ISO 9660 逻辑文件系统必须知道在写会话的开始哪些文件要写入。需要这些信息来创建指向光盘上文件物理位置的指针。ISO 13346 使用包写入技术允许在任何时候将文件添加到 CD-R 和 CR-RW 光盘中。在每个包写入会话结束时，虚拟分配表（virtual allocation table）被添加到光盘以描述每个文件的物理位置。每个新创建的虚拟分配表都包括以前的表。

CD-RW 驱动器中的激光有 3 种功率。在读操作时，它提供了检测凹坑边缘的光束，在

最低功率模式下运行。在写操作过程中，对记录层表面进行加热以创建非晶态时，激光在最高功率模式下运行。写功率将表面局部加热到大约 600℃。当其快速冷却时，液体冻结和收缩将创建一个凹坑。

当 CD-RW 驱动器擦除数据时，激光在比写操作低的功率模式下运行，激光对表面充分加热到数据层转化为晶态。相变材料加热到 200℃，变为晶态，原子成为有序状态。为保证继续提供光学可区别的状态，材料可以允许的写和擦除数量是有限的。CD-RW 仍然是一种多数时间为读的介质，而不是完全意义上的读 / 写介质。

2. 磁光存储

与相变技术相对应的另一个方法是磁光（magneto-optical，MO）记录。MO 系统不能与传统的 CD 驱动器完全兼容，低成本的 CD-RW 驱动器的兴起导致 MO 技术的没落。

回忆前面讲的居里温度，它定义了磁性材料退磁的温度。有些物质的居里点有 200℃，这意味着它们可以通过激光加热而退磁。图 3-48 说明了 MO 系统的原理。携带数据的盘面由具有低居里点的铁磁材料构成。在正常操作中，表面磁场被磁化为盘面的垂直方向。如果表面被激光加热，它将会退磁。然而，由于盘面下有一个电磁铁，当表面冷却时将导致磁场翻转。

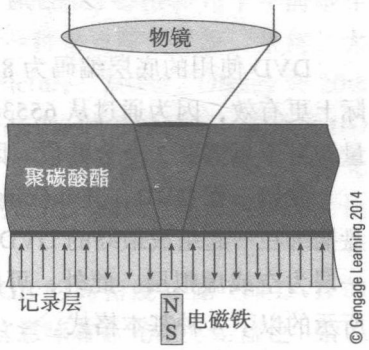


图 3-48 磁光记录

MO 盘可以通过磁性和光之间的互动进行读取。当偏振光通过磁性材料，将会改变光的偏振。这种现象被称为克尔效应（Kerr effect）。

类似 CD 的光学系统可以用来读取 MO 光盘。当光线从盘片反射回来时，如果表面被磁化其偏振面将旋转约 0.5°，而如果表面磁化翻转则其偏振面将向另一个方向旋转 0.5°。因此，两种磁状态反射回的光的偏振面有 1° 的差异。光学系统和传感器能够检测到这种差异。

3.7.5 DVD

DVD 是 CD 的创新发展产物，它的意思是数字视频光盘（digital video disc）或数字通用光盘（digital versatile disc）；似乎没有人知道哪一个才是正确的。重要的是经过优化后 CD 可以存储约 75min 高质量的音频，而 DVD 可以存储一部电影。不同于 CD-ROM，DVD 具有几种容量，这取决于它具有一个还是多个数据记录层。DVD 技术在 20 世纪 90 年代初由 Toshiba、Time Warner、Sony 和 Philips 等公司研发。

一些 DVD 技术的发展与好莱坞有密切的联系，好莱坞强烈影响着新兴标准。特别是，DVD 设计用来存储 133min 的视频信息编码（足以覆盖大多数主流电影）。DVD 提供了高质量的视频和音频，包括 3 个独立的声道，使得相同的 DVD 可以被不同国籍的观众观看。声道中还包括杜比降噪编码，可以作为家庭影院（home theater）技术的基础，杜比编码大约在 1991 年开始出现。

DVD 技术和 CD 技术几乎是一样的。事实上，可以说 DVD 就是拥有发展并成熟了 10 年以上技术的 CD。DVD 光盘的凹坑更加紧密，最小凹坑的大小是 0.4μm，而不是 CD 使用的 0.83μm。激光波长从 780nm 减少到 640nm。同样，道间距从 1.6μm 减少到 0.74μm。图 3-49 说明了 CD 和 DVD 光轨的结构。

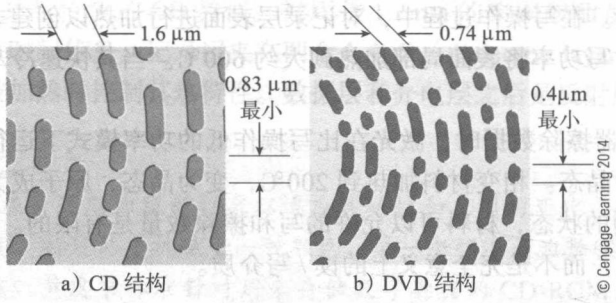


图 3-49 CD 与 DVD 结构

DVD 使用的底层编码为 8 ～ 16 位编码而不是 CD 使用的 8 ～ 14 位编码。DVD 格式实际上更有效，因为通过从 65536 个模式中选择 256 个模式，可以在记录的数据中避免直流分量，且不再需要 3 个合并位。因此，每个字节只需要 16 位而 CD 需要 17 位。

先做一张 DVD，其 0.6mm 的厚度比 CD 薄一半。再做第二张盘，也是 0.6mm 厚，两张盘捆绑在一起组成最终的 DVD。该 DVD 有两个 0.6mm 的层，其中一层为 DVD 固有层，另一层为空或虚拟层。此外，可以在同一张盘片上具有两个不同的数据层。DVD 支持图 3-50 所示的以下 4 种基本格式。

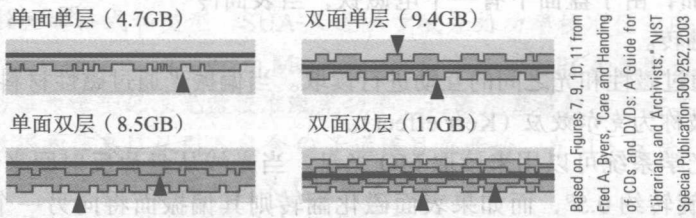


图 3-50 多层多面 DVD 结构

- DVD-5 为单面单层盘。总容量 4.7GB。
- DVD-9 为单面双层盘。总容量 8.5GB。
- DVD-10 为双面单层盘。总容量 9.4GB。
- DVD-18 为双面双层盘。总容量 17.0GB。

可记录 DVD

可记录 DVD 随着 DVD 的发展而发展，就像 CD 跟着 CD-ROM 的发展一样。然而，DVD 技术并不像 CD 技术那样迅速稳定下来，CD、CD-R 和 CD-RW 依次出现并具有合理的向后兼容性。几年之内，DVD 后出现了 DVD-RAM、DVD-RW、DVD+RW 和 DVD-R。

DVD-R 是一次写的介质，其容量为 4.7GB 或 9.4GB，可以在大多数兼容的 DVD 驱动器中使用。它首次出现在 1997 年，是一种低容量的 3.95GB 版本。1998 年，出现了第一种可重写的设备称为 DVD-RAM，它依赖相变和磁光技术向光盘写入数据。第一代 DVD-RAM 具有 2GB 的容量，但到 1999 年增加到 4.7GB。该系统与其他 DVD 格式不兼容。

Pioneer 研发了 DVD-RW 系统，其目标是创建与 DVD 和 DVD-R 兼容的格式。这种格式，首先出现在 1999 年，使用相变技术来读、写和擦除信息。650nm 波长的激光加热相变合金晶体使其从晶态转变为非晶态，这与 CD-RW 系统完全一样。另一种可重写的 DVD 出现在 2002 年，被命名为 DVD+RW 格式。它自称比其竞争对手 DVD-RW 更加兼容。

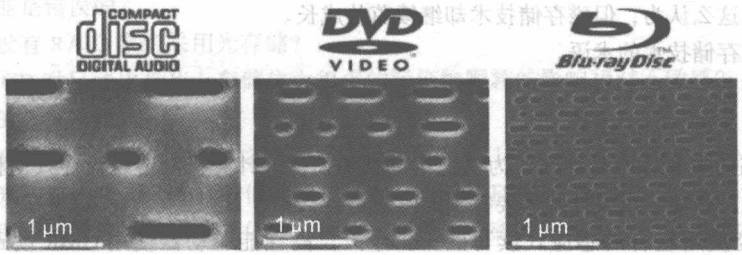
3.7.6 蓝光

就像DVD取代CD一样，蓝光技术正在取代DVD。蓝光的引入受高清电视（high defined television，HDTV）的驱动，HDTV需要比DVD提供更大的容量。没有新的存储介质，高清家庭影院是不可能的（不考虑有线广播这种情况）。针对提高光记录介质容量的问题提出了两种不同的解决方案：HD DVD和蓝光。两种系统都在发展，每种技术背后都有媒体巨头的支持。蓝光由Sony、Panasonic、Philips、LG、Pioneer和Apple倡导。HD DVD由Toshiba、Hitachi、Microsoft和NEC倡导。

这两种不兼容的格式似乎是关于VCR录影带的VHS和Betamax标准在几十年前争斗的一种回声。两种格式迫使消费者做出选择并使商店使用某一种格式存储电影。最终，大制片厂在标准大战中具有更强的讨价还价的能力。Sony Picture、MGM、Disney和20th Century Fox等制片厂选择了蓝光，只有Universal Studio（约占9%的市场份额）选择了HD DVD。Sony为其流行的PlayStation 3游戏机（仅在美国就有320万台游戏机）选择了蓝光以增加蓝光产品的需求。HD DVD的另一个葬送者是沃尔玛的促销。因此，蓝光胜利并开始流行了，世界避免了一场持久的格式大战。

通过使用波长为405nm的高频激光，蓝光可以达到25GB的高密度存储（即DVD的5.3倍）。可见光谱从620nm（红色）延伸到450nm（紫色），这意味着蓝光激光是蓝色/紫色的，故而得名。DVD激光波长为650nm（红色），CD激光波长为780nm，为红外光谱。蓝光光盘与CD和DVD不同，因为蓝光的数据层位于仅有0.1mm的封面保护层下方。在蓝光系统中聚焦光束使用镜头的数值孔径（0.85）要高于DVD（0.6）和CD（0.45）。回忆前面介绍的，高数值孔径允许更小的光点尺寸。

像DVD一样，蓝光支持双层结构，可以提供50GB的容量。图3-51提供了三代记录介质的图像来展示它们的相对密度。



Ferry Zijp, "Near-Field Optical Data Storage," Philips Electronics, 2007. Courtesy of Philips.

图3-51 三代记录光盘的扫描电镜图像

蓝光和DVD之间的显著区别是盘片的结构和记录层的位置。在DVD中，数据记录层夹在盘片中间（CD的数据记录层位于其底部）。在蓝光介质中，数据记录层在1.1mm厚衬底的顶部且被0.1mm厚的保护层覆盖。将数据层放在盘片顶部附近的好处是增加了盘片对倾斜（tilt）的容忍。并不是所有的盘片都是完全平坦的（即对激光来说是正常的），所以会有一定程度的倾斜。蓝光光盘能够容忍的倾斜角度上限为0.64°。

DVD通过两个盘片来生产，将数据层放在一个盘片上然后将其夹在两个盘片中间。蓝光光盘更容易制造，因为数据层放在盘片顶部然后加上一个保护层。该保护层比CD和DVD的聚碳酸酯衬底更加坚硬，更好地防止光磨损（light abrasion）。现代蓝光光盘可能具有复杂的涂层结构。例如，Verbatim盘在数据层上具有硬模层、覆盖层和保护层。在数据层

的下方还有另外的保护层、反射层，然后才是聚碳酸酯衬底。

本章小结

本章介绍了二级存储系统。这些系统能够以非常低的成本保存海量的数据。不幸的是，二级存储系统的访问时间比半导体存储器的要长 10^6 量级，这意味着计算机必须使用有效的内存管理系统来隐藏二级存储设备的延迟。一些二级存储的容量在以令人震惊的速度增长，在 21 世纪初达到每年增长 60% 左右。虽然二级存储系统的速度经过了多年的改进，但其访问时间缩短得很慢，除非有技术上的根本性改变，人们不太可能看到任何真正的进展。

主要的二级存储技术为磁存储和光存储。这些技术都占据着计算机世界中的一席之地。磁存储比光存储提供了更好的访问时间、更大的容量以及更好的读 / 写性能。这些特点使得磁存储对于计算机运行时存储程序和数据的驱动器来说十分理想。光存储系统提供低成本的移动存储介质，主要包括 CD（600MB）、DVD（约 4.7GB）和蓝光（25GB）。

本章也介绍了如何使用硬盘来构建低成本、大容量存储系统 RAID，它比单个硬盘具有更高的可靠性和更好的性能。本章还引入了固态盘技术，它正在取代旋转的磁盘驱动器，但也具有其自身的局限性。

磁存储是一种成熟的技术。主要在磁盘容量和接口方面进行不断更新。光存储目前正处于蓝光技术取代 DVD 和 CD 的阶段。然而，光技术的进步并不是主要由 PC 世界驱动，而是由好莱坞和其他电影制片方推动，因为它主要是作为存储音像制品的载体。

习题

3.1 是什么限制了硬盘的面密度？

3.2 本章参考了 2000 年 12 月在《IEEE Spectrum》上发表的一篇关于磁存储的论文，标题为“Magnetic Storage: The medium that wouldn't die”。该标题暗示磁存储技术应该早就过时了。请解释为什么许多人这么认为，但磁存储技术却继续茁壮成长。

3.3 解释下列用于磁存储技术的术语。

a. 磁场	b. 磁导率	c. 磁致伸缩	d. 剩磁场
e. 铁磁体	f. 居里点	g. 硬磁存储	h. 软磁存储

3.4 硬盘设计师面临的主要障碍是什么？为笔记本电脑和便携式计算机制造硬盘导致工程师需要一组不同的设计标准。移动硬盘设计师需要特别的考虑是什么？

3.5 硬盘驱动器曾经使用组合的感应式读写磁头。今天，高性能驱动器使用 GMR 读磁头。什么是 GMR 读磁头？将读写磁头分离的优势是什么？

3.6 为什么难以从如磁带或磁盘这样的磁介质中完全擦除数据？为什么数据在被写入磁盘之前要进行编码？

3.7 下面的数据序列是在磁盘中记录的 MFM 编码。画出写磁头看到的结果波形。

0101001100001111

3.8 决定扇区最佳大小的标准是什么？这些标准是永久不变的还是会随时间变化？在计算世界中哪些发展可能会影响这个问题的答案？

3.9 硬盘驱动器的道密度为 524 000tpi，面密度为 29.7Gb/in^2。磁道上的位密度是多少？

3.10 假设某磁盘驱动器有以下参数。短寻道的时间为 $1+0.2\sqrt{n}$ ms，长寻道的时间为 $3+0.003n$ms。假定短寻道小于 200 条磁道。假设驱动器顺序访问以下磁道，所需的近似寻道时间（忽略旋转延迟）是多少？

2344, 2345, 2347, 2310, 1566, 1580, 2350, 2352, 1210, 3230

3.11 假设问题 3.10 中的磁道访问顺序从最小磁道号开始以递增顺序至最大磁道号，寻道时间又会是多少？

3.12 如何使磁盘驱动器更快（即制造商可以有哪些选项）？

3.13 某硬盘具有以下参数：6 面，每面 2 万条磁道，每条磁道 256 个扇区，每个扇区 512 字节，转速为 7200r/min。计算：磁盘的容量、旋转延迟、定位扇区后读取数据的速率。

3.14 描述以下磁头调度算法：FIFO、SSTF、SCAN 和 C-SCAN。

3.15 某制造商希望生产旋转延迟为 1ms 的磁盘，需要的转速是多少？

3.16 某磁盘驱动器具有以下特性：转速为 7200r/min；每条磁道 256 个扇区；寻道间距小于 10 条磁道时跨越每条磁道的时间为 2ms；寻道间距小于 20 条磁道时跨越每条磁道的时间为 1ms；寻道间距小于 40 条磁道时跨越每条磁道的时间为 0.4ms。假定后续扇区与当前扇区位于同一条磁道时不需要寻道，要访问以下扇区所需的近似时间为多少？每一对数据中前者表示磁道号，后者表示扇区号。

(10,16), (10,38), (10,50), (10,51), (11,101), (9,119), (24,36), (24,35), (198,40)

3.17 SMART 技术是什么，它是如何帮助硬盘用户的？

3.18 最流行（即商业上最成功）的 RAID 为 RAID 1 和 RAID 5，为什么？

3.19 假设磁盘驱动器有 5000 个柱面，编号为 0 ～ 4999。驱动器当前正在服务对 143 号柱面的请求，前一个请求是柱面 125。以 FIFO 排队的请求顺序为：86、1470、913、1774、948、1509、1022、1750 和 130。从当前磁头位置开始，按照以下磁盘调度算法，磁臂完成这些请求需要移动的总距离（以柱面表示）是多少？

a. FCFS；b. SSTF；c. SCAN

3.20 如果单个驱动器失效的概率是每 1000 小时 0.000 001 次，由 4 个磁盘组成的阵列中只有一个磁盘失效的概率是多少？两个磁盘一组，由 4 组构成的阵列中只有一个磁盘失效的概率是多少？

3.21 当计算类似 RAID 这样有多个磁盘的系统的可靠性时，概念上显然有，单个磁盘失效的概率为 1/1000，则两个磁盘失效的概率为 1/100 万，这是由于不相关的概率相乘的结果。为什么说这种说法可能是错误的？

3.22 为什么没有 RAID 系统采用光存储？

3.23 为什么 CD 对尺寸比盘片上存储位大很多的污染物颗粒的影响相对不敏感？为什么需要激光器将光盘上的数据读出？

3.24 为什么在将数据位记录到 CD 之前需要进行类似 8 ～ 14 位这样的编码？

3.25 如果激光的波长为 780nm，数值孔径为 0.45，照在 CD 表面光点的大小是多少？

3.26 恒角速度和恒线速度的区别是什么？为什么一种用于磁盘而另一种用于光盘？

3.27 磁盘转速为 7200r/min，磁头位于离盘片中心 2in 的磁道上，相应的线速度和角速度为多少？

3.28 可写 CD 是如何实现的？

3.29 如果 20 世纪 80 年代前的磁带为 1/2in 宽，在 9 条磁道上以 128 字符 /in 的密度记录数据，用位表示的磁带每平方英寸的面密度是多少？该面密度与现代磁盘面密度的比例为多少？

3.30 1996 年，NEC 发布了 64MB 的闪存，芯片尺寸为 98mm^2。用 b/in^2 表示的话，相应的面密度为多少？

3.31 到 2002 年，已经有了尺寸为 125mm^2、容量为 1Gb 的闪存。用 b/in^2 表示的话，相应的面密度为多少？

3.32 2010 年 9 月，Toshiba 发布了第一款嵌入式闪存模块（即嵌入电路板而非直插模块）。这些模块的容量为 128GB，FBGA 封装尺寸为 17mm × 22mm × 1.4mm。用 b/in^2 表示的话，相应的面密度为多少？

3.33 若将问题 3.32 中的嵌入式闪存模块制成标准的 3.5 英寸磁盘驱动器，假定一半的体积要用于散热

和接口电路，该驱动器可存储的数据量为多少？

3.34 如果蓝光激光的波长为405nm，物镜的数值孔径为0.85，蓝光激光的最小光点大小是多少？

3.35 如果DVD激光的波长为650nm，物镜头的数值孔径为0.6，DVD激光的最小光点大小是多少？

3.36 假定一个光点可以烧出一个1或0，激光的波长为405nm，物镜的数值孔径为0.85，光道内径和外径分别为23mm和59mm，计算单层蓝光光盘的理论最大容量。

3.37 对比闪存和硬盘的优点和不足？什么是耗损平衡，为什么要在闪存中使用它？

3.38 某硬盘由线性膨胀系数为1.5×10^{-6}/℃的材料制成。磁道间距为1000nm。假定磁头精确定位于半径为2in的磁道上方，磁头偏离磁道20%将使错误率增加。该磁盘可以忍受的温度提升为多少度？从答案中可以得到什么结论？

第 4 章

Computer Organization and Architecture: Themes and Variations

输入 / 输出

“我应该是个竞争者。”

——Marlon Brando，《码头风云》

“永远不要把编译时该做的事情推迟到运行时。”

——David Gries

“任何人都可以制造快速的 CPU。关键是如何制造快速的系统。”

——Seymour Cray

“这虽是疯话，却有深意在内。”

——莎士比亚，《哈姆雷特》

“若一次撒谎，则总是撒谎。”

——罗马法律条文

“除了死亡、税收，没有什么是必然的。今天，USB 显然在不断取得成功。”

——Cheryl Coupé

在计算机可以处理和存储信息之前，它们必须从外部世界读取数据。同样，计算机必须能够将信息传输给外部设备。接下来将介绍数据是如何进入和移出计算机的，以及这些数据是如何通过总线家族（family of bus）在计算机各功能部件之间进行传输的。

计算机不只是一整块硅片，它由互相连接的子系统构成，例如存储器阵列、磁盘驱动器、视频显示系统以及一个或多个处理器。每个子系统执行特定的任务并与计算机中其他子系统进行通信。本书关注的是内部子系统之间以及系统与外部之间是如何进行通信的。从 I/O 是不是计算机体系结构或计算机组成的一部分这个问题开始。I/O 基本上属于计算机组成（organization）的范畴，因为它关心的是数据如何从一个地方传输到另一个地方。一些计算机体系结构的教材集中讨论处理器 ISA 的细节或处理器内部的组织，对总线和 I/O 讨论较少。本章将更深入地研究 I/O，因为它对系统性能的贡献与 CPU 一样重要。此外，对 I/O 的研究还包括了理解数字系统必不可少的重要主题（例如，缓冲、握手以及协议）。

本章首先概述了 I/O 操作和数据从一个位置移动到另一个位置的方式。然后考察了从 PCI 总线到 USB 接口的 I/O 系统。4.1 节介绍计算机是如何处理 I/O 事务的。使用事务（transaction）这个术语是因为 I/O 包括处理器与输入 / 输出设备之间的对话。

本章的主题主要是总线（bus），数据利用它在计算机内部以及计算机与其外围设备之间进行移动。如果连接 CPU 和存储器的总线不能以 CPU 所需的速度提供数据，将 CPU 和存储器做得再快也没有意义。现代个人计算机的发展归功于处理器的发展，也同样归功于总线技术的发展。人们喜欢性能，但其购买的是各种功能。事实是，用户可以将计算机与互联

网、数码相机、多个打印机、扫描仪、MP3 播放器以及 iPad 相连接，使个人计算机如此值得拥有。如果用户不能轻易地通过一条或多条总线将计算机连接到以上所有这些外部系统的话，它将永远只是一台文字处理器、数据库引擎或者游戏机。

今天，人们把总线说成是一种*体系结构和系统基础设施*，这是因为总线不再是将计算机内两个或多个部分联系在一起的简单信号通路。多年来，总线不断地发展，提供了比以前更多的功能。例如，在多处理环境中，总线不得不应对多个 CPU。此外，现代高性能计算机系统不止一条总线；它有层次化总线——每条总线都为特定的目的进行了优化。一些总线在 CPU 和随机存取存储器之间提供高速数据传输，而其他总线以更慢的速率、更长的路径向打印机和调制解调器发送数据。

4.1 I/O 的基本原理

下面介绍 I/O 事务的基本原理与相关词汇。图 4-1 描述了一个通用的计算机系统，它具有 CPU、I/O 控制器、外围设备，以及将 CPU 与存储器和外围设备连接在一起的系统总线。在图 4-1 中外围（peripheral）一词出现了两次，它既用来描述外部设备（如连接到计算机的打印机或鼠标），又描述外部设备与 CPU 之间的接口。

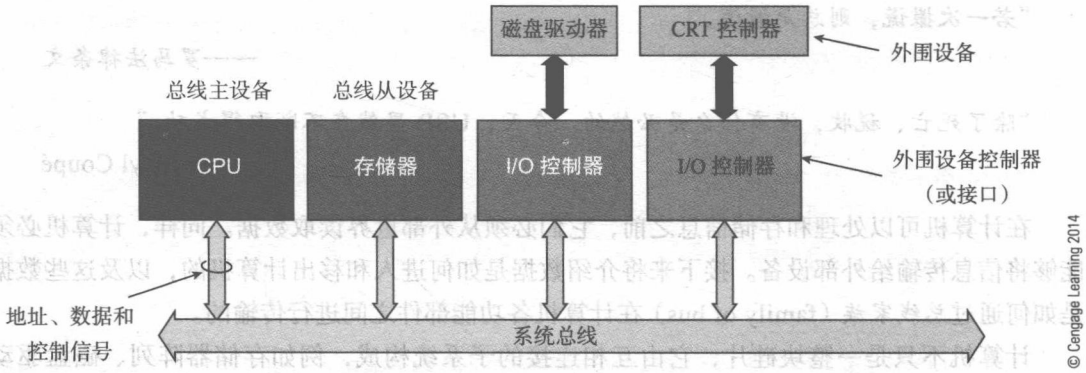


图 4-1 处理器、总线与 I/O 系统

图 4-2 从另一个角度来观察计算机系统的层次结构。CPU 和存储器位于系统的核心。将处理器和存储器连接到外部设备的外围接口（peripheral interface）显示在两个框中：一个包含内部（internal）外设（如磁盘驱动器），一个包含外部（external）外设（例如调制解调器、打印机和扫描仪）。内部外设和外部外设没有根本的区别。

I/O 控制器可以被认为就是一种*协议转换器*（protocol converter），因为它既符合计算机总线协议的需要，也满足外部外围设备协议的需要。它可以进行不同数据格式之间的转换（例如改变电压水平，对信号进行编码，将并行数据转换为串行数据，等等）。I/O 控制器的复杂性可以和 CPU 的复杂性相匹敌。

一些计算机使用特殊的机器指令和控制信号来处理 I/O 事务。例如，一个假想的微处理器可能通过以下指令将寄存器 r3 中的一个字节发送给称为通道 5 的磁盘驱动器：

```
OUTPUT r3,Chan5      ; 通过通道 5 将 r3 中的数据发送到磁盘
```

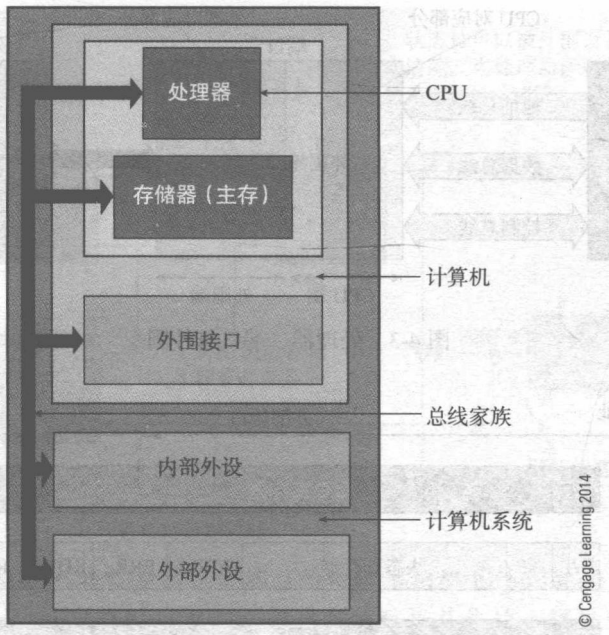


图 4-2 计算机系统的层次

Intel 处理器的指令集中有一组 I/O 指令，允许 8080 和 8086 系列成员之间通过寄存器和 8 位的端口传输数据。例如，指令 `IN AL, 4CH` 将端口地址为 $4C_{16}$ 的一个字节传输至寄存器 AL，指令 `OUT 4EH, AL` 将寄存器 AL 中的一个字节传输至端口地址 $4E_{16}$。处理器产生控制信号，使 I/O 控制器可以检测到访问。因为没有必要，所以很少有微处理器采用这种方法。

具有专用 I/O 指令的处理器使用地址总线来定义数据端口，在写周期中通过该端口发送数据，在读周期中通过该端口获得数据。这种处理器被认为既具有*存储空间*（memory space）又具有 I/O 空间（I/O space）。专用 I/O 体系结构，例如 8080 结构，需要特定的硬件并消耗了指令空间（每条执行 I/O 操作的指令本可以做其他事情）。

存储器映射的外围设备

微处理器可以不需要 I/O 功能，因为 I/O 事务和存储器访问没有本质区别。在处理器看来，向外围设备输出一个字与在存储器中存储一个字完全相同，而从外围设备获取一个字与从存储器中读取一个字也完全相同。将 I/O 事务看作存储器访问被称为*存储器映射*（memory mapped）的 I/O。该方式可以像访问存储器那样访问 I/O，但这并不意味着可以忽略 I/O，因为随机访问存储器的性质与典型的 I/O 系统的性质完全不同。当实现 I/O 结构时，必须考虑 I/O 设备本身的特点。例如，当向磁盘驱动器写入一个文件时，可能会每隔几 μs 就发送一个新的数据字节。图 4-3 显示了典型的存储器映射 I/O 端口（*外围接口芯片*）看起来与处理器非常相像。

对于主机 CPU 来说，外围设备看上去就像连续的存储器地址，如图 4-4 所示。图 4-3 中外围接口芯片左手边部分在 CPU 看起来完全像一个存储器部件。外围接口芯片的另一半，即右手边部分是执行接口所需特定操作的*外围端*（peripheral side）。例如，磁盘控制器接口可能需要寻找特定的扇区，或者串行接口芯片可能把一个字节转换成可以通过单根信号线传输的脉冲序列。外围接口芯片与适当的外围设备（例如，磁盘驱动器、鼠标、键盘或打印机）连接。

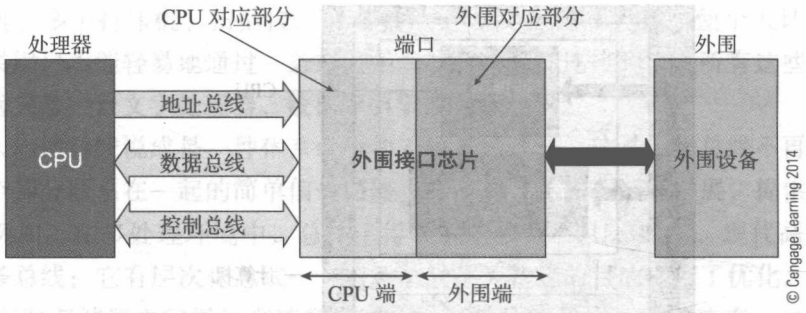


图 4-3 处理器、端口与外围

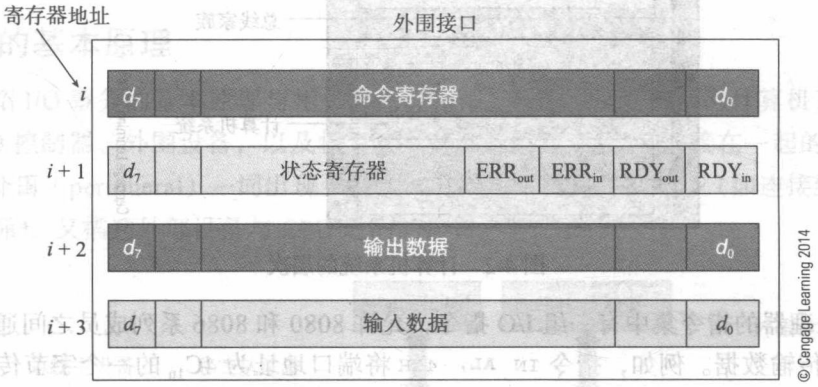


图 4-4 存储器映射的寄存器

图 4-4 中的存储器映射端口具有连续的 4 个寄存器，其地址为 i、i+1、i+2 和 i+3。假定外围设备为 8 位的设备，每个连续的地址对应一个字节。在具有 32 位数据总线的系统中，寄存器的地址将为 i、i+4、i+8 和 i+12。地址 i 处包含一个命令寄存器，它定义了操作模式和外围设备的特性。大多数存储器映射 I/O 端口根据特定的应用被配置成可以执行多种模式的操作。通过提供多功能的 I/O 设备，半导体制造商使用单个芯片就可以覆盖大部分的市场需求。

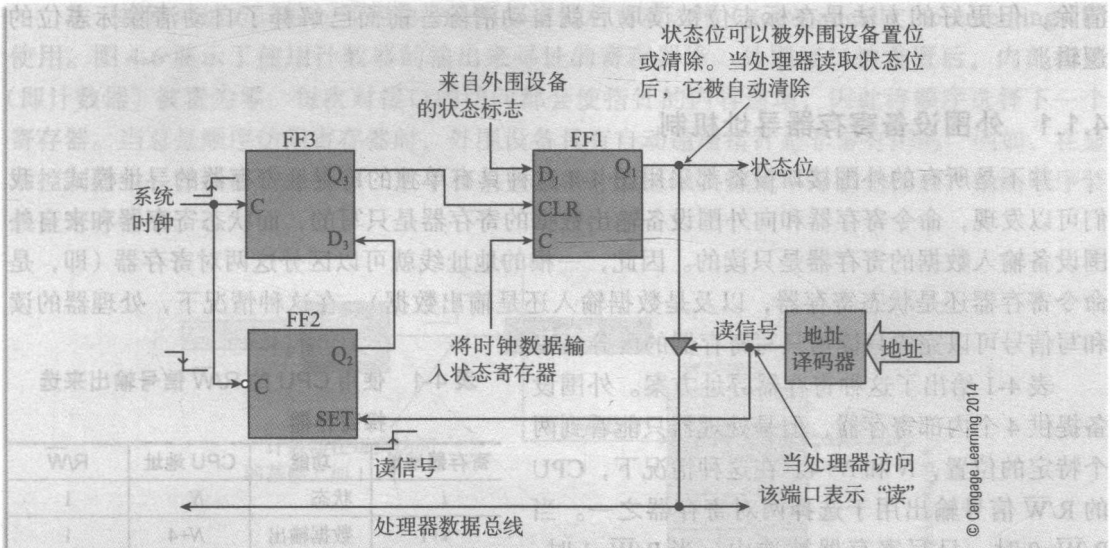
第二个位置为地址 i+1，包含了端口的状态，它由相应的外围设备设置。处理器可以读取状态信息来确定参加数据事务处理的端口是否已经准备就绪，或者是否发生了错误。例如，连接到存储器映射 I/O 端口的打印机需要设置错误位来表示打印纸用完了。在图 4-4 的这个例子中，构建了通用的状态位[⊖]，例如 ERR_{out}、ERR_{in}、RDY_{out} 和 RDY_{in}（分别表示输出错误、输入错误、输出就绪和输入就绪）。

自动清除标志位

下图说明了自动清除机制可能的结构。当外围设备响应了一个事件向 D_1 输入信号，使触发器 FF1 的输出端 Q_1 置位。

假设外围设备已经产生了数据，并使 FF1 锁住数据的状态。此时，Q_1 包含了数据位（即状态标志位），可以被处理器读取。

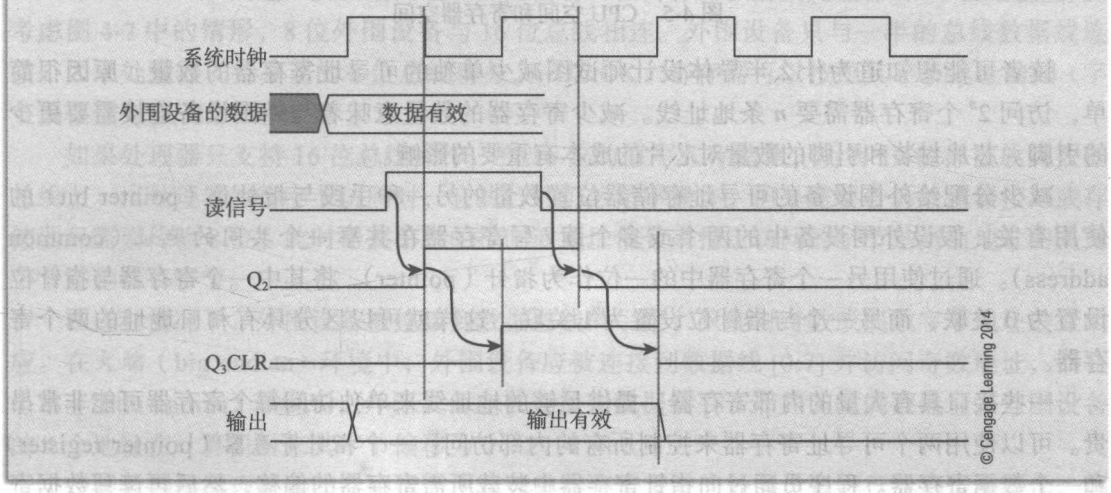
⊖ 在条件码寄存器中，每位状态位通常指的就是一个状态标志位。



处理器通过生成适当的存储器映射端口的地址来读取 FF1 的状态。有效的地址将三态门使能，并将 Q_1 的输出放到处理器可以读取的数据总线的一根信号线上。读信号同时还在下个时钟的下降沿将触发器 FF2 置位。

FF2 的输出 Q_2 与 FF3 的输入 D_3 连接在一起，使得 Q_3 将在下个时钟的上升沿有效。由于 FF3 的输出连接到触发器 FF1 复位（CLR）信号的输入，当 Q_3 为高电平有效时状态标志被自动清除。当读信号变低时，触发器 FF2 和 FF3 回到其正常的低电平不活动状态，CLR 信号从 FF1 上消除。

下图给出了该电路的时序图。自动清除电路必须谨慎使用，因为不经意的读操作就会使该标志位被清除。该情况在某些特定的情况下会引起副作用。例如，某些处理器在访问地址 l 的时候向地址 l+1 进行了虚拟访问。



地址 i+2 和 i+3 分别用来保存向外围设备输出以及从外围设备接收的数据。正是通过这些地址，处理器可以实现与 I/O 端口的通信。

一些外围设备具有可以自动清除的状态标志位。假设某个标志位，通过 I/O 事件（event）（例如数据就绪、设备忙或者数据错误）置位。一旦读取该标志后，虽然可以手动

清除，但更好的方法是在标志位被读取后就自动清除。前面已解释了自动清除标志位的逻辑。

4.1.1 外围设备寄存器寻址机制

并不是所有的外围接口设备都采用图 4-4 这种具有单独的可寻址寄存器的寻址模式。我们可以发现，命令寄存器和向外围设备输出数据的寄存器是只写的，而状态寄存器和来自外围设备输入数据的寄存器是只读的。因此，一根的地址线就可以区分这两对寄存器（即，是命令寄存器还是状态寄存器，以及是数据输入还是输出数据）。在这种情况下，处理器的读和写信号可以完成只读与只写寄存器的区分。

表 4-1 给出了这种寄存器寻址方案。外围设备提供 4 个内部寄存器，但是处理器只能看到两个特定的位置，*N* 和 *N*+4。在这种情况下，CPU 的 R/$\overline{W}$ 信号输出用于选择两对寄存器之一。当 R/$\overline{W}$=0 时，只写寄存器被选中；当 R/$\overline{W}$=1 时，只读寄存器被选中。图 4-5 强调了外围设备寄存器空间被分为只读和只写区域的方式。

表 4-1 使用 CPU 的 R/$\overline{W}$ 信号输出来选择寄存器

寄存器地址	功能	CPU 地址	R/$\overline{W}$
i	状态	*N*	1
i+1	数据输出	*N*+4	1
i+2	控制	*N*	0
i+3	数据输入	*N*+4	0

有些计算机分别有读和写信号。有些计算机只有一个 R/$\overline{W}$（读 / 非写）信号，在读操作时为高电平，在写操作时为低电平。

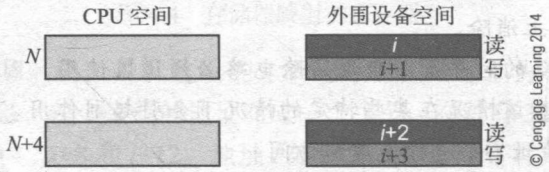


图 4-5 CPU 空间和寄存器空间

读者可能想知道为什么半导体设计师试图减少单独的可寻址寄存器的数量。原因很简单，访问 2^n 个寄存器需要 *n* 条地址线。减少寄存器的数量意味着与外围设备连接需要更少的引脚。芯片封装和引脚的数量对芯片的成本有重要的影响。

减少分配给外围设备的可寻址存储器位置数量的另一种手段与指针位（pointer bit）的使用有关。假设外围设备中的两个或多个读 / 写寄存器在共享一个共同的地址（common address）。通过使用另一个寄存器中的一位作为指针（pointer），将其中一个寄存器与指针位设置为 0 关联，而另一个与指针位设置为 1 关联，这样就可以区分具有相同地址的两个寄存器。

一些接口具有大量的内部寄存器，提供足够的地址线来单独访问每个寄存器可能非常昂贵。可以使用两个可寻址寄存器来控制所有的内部访问：一个指针寄存器（pointer register）和一个数据寄存器。程序员通过向指针寄存器中装载所需寄存器的偏移，然后再读写数据寄存器来实现对外围设备内部寄存器的访问。

这种技术只需要一根地址线来区分内部寄存器，但会造成访问速率的降低。在外围设备内部实现基于指针的寻址方式性价比较高。例如显示控制器，其配置寄存器在初始化后就很少被访问。

基于指针寻址方式的一种变化涉及内部指针对自动递增（automatically incrementing）的使用。图 4-6 展示了使用计数器的输出来寻址的寄存器堆。外围接口被重置后，内部指针（即计数器）被置为零。每次对接口的访问都会使指针的内容递增，因此将顺序选择下一个寄存器。当总是顺序访问寄存器时，外围设备具有自动递增指针是非常有用的。例如，在显示控制器中，显示器的分辨率、水平和垂直像素计数器以及帧速率等总是按照严格的顺序装载至寄存器中。

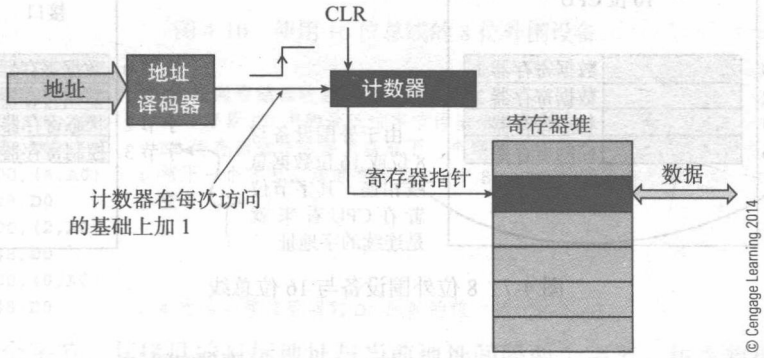


图 4-6 通过自动递增指针来访问多个寄存器

4.1.2 外围设备访问和总线宽度

存储器映射的外围设备需要考虑的第二个因素是外围设备数据总线宽度与主机数据总线宽度之间的关系。许多外围设备通过 8 位宽度的总线连接到 32 位的计算机。

当 8 位外围设备通过 8 位数据总线与 8 位处理器连接，或当 16 位外围设备通过 16 位数据总线与 16 位处理器连接，这都是很容易的。当低成本的 8 位外围设备与 16 位或 32 位的总线相连，事情就变得复杂。将 8 位外围设备与 16 位总线相连会出现两个问题：存储端格式（endianism）问题以及映射问题，即如何将 8 位寄存器映射到处理器的 16 位地址空间。考虑图 4-7 中的情形，8 位外围设备与 16 位总线相连。外围设备只与一半的总线数据线连接。如果处理器支持真正的 8 位总线事务，一切 OK，寄存器可以按照其字节地址访问（字节偏移量为 0、1、2 和 3）。

如果处理器只支持 16 位总线操作，当 16 位的值写入存储器，所有 16 位都被放到数据总线上。当处理器执行字节访问时，它执行的仍然是字操作，但是需要告诉处理器接口或存储器只需要传输 8 位。此时需要单独的控制或地址信号来指定访问的字节是当前地址的高位字节还是低位字节。

在这种情况下，外围设备被连接到数据总线的一半，只能对奇数或偶数字节地址进行响应。在大端（big endian）环境中，外围设备应被连接到数据线 [0:7] 并访问奇数地址，而在小端（little endian）环境中，外围设备应被连接到数据线 [8:15] 并访问偶数地址。外围设备的 4 个地址在计算机看来其字节偏移为 0、2、4 和 6。

一些处理器实现了某些专用指令用于与具有字节宽度的外围设备进行数据传输操作。例如，32 位的 68000 处理器具有 MOVEP 指令（其含义为移动外围设备），该指令可以从 8 位存储器映射的外围设备读写 16 位或 32 位的数据。图 4-8 显示了具有 4 个内部寄存器的外围设备以及 CPU 地址图，其中外围设备的数据空间在大端模式处理器存储器空间中被映射到连续的奇地址。

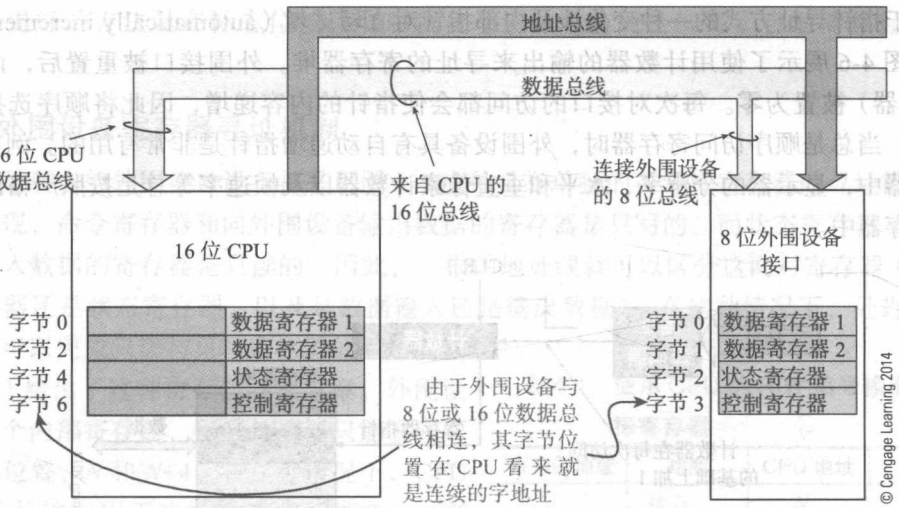


图 4-7 8 位外围设备与 16 位总线

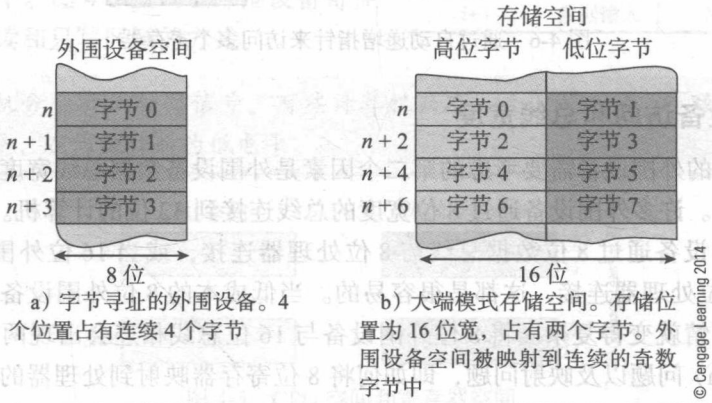


图 4-8 将 8 位外围设备映射到 16 位总线

图 4-9 显示的外围设备具有 4 个 8 位寄存器，以存储器映射方式映射至地址 0x080001。在程序员看来，4 个 8 位寄存器的地址为 0x080001、0x080003、0x080005 和 0x080007。地址 0x080000、0x080002、0x080004 和 0x080006 不存在，无法访问。MOVEP 指令自动在数据寄存器以及具有字节宽度的存储器映射外围设备间移动 16 位或 32 位的值。寄存器的内容被移动到连续的偶数（或奇数）字节地址。例如，MOVEP.L D2, (A0) 指令将寄存器 D2 中的 4 个字节拷贝至地址 [A0]+0、[A0]+2、[A0]+4 和 [A0]+6，其中 A0 是一个地址或指针寄存器。

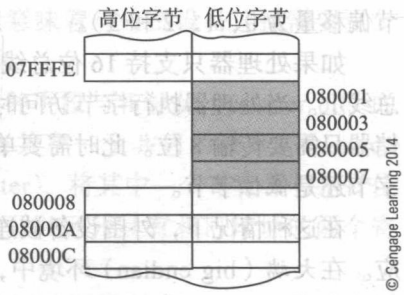


图 4-9 具有字节宽度的存储器映射的外围设备实例

图 4-10 演示了 MOVEP.L D0, (A0) 指令从地址 0x080001 开始如何将寄存器 D0 中的 4 个字节拷贝至存储器中连续的奇数地址。68000 处理器中代码后缀“.L”代表 32 位操作，“.B”代表字节操作。数据寄存器中的最高位字节被传输给最低的地址。若没有 MOVEP 指令，则需要通过下面的代码将 4 个字节传输至存储器映射的外围设备。

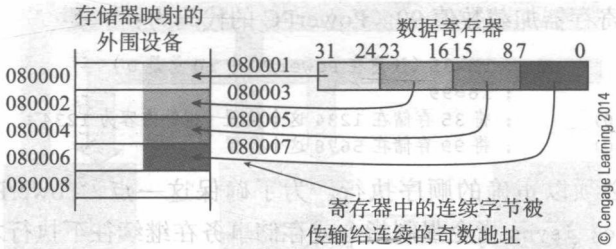


图 4-10 使用 16 位总线的 8 位外围设备

```
MOVE.L  #Peri,A0      ; A0 指向存储器映射的外围设备
MOVE.B  D0,(6,A0)     ; 将寄存器 D0 中的最低位字节传输给外围设备
ROR.L   #8,D0         ; D0 寄存器循环右移来获取下一个字节
MOVE.B  D0,(4,A0)     ; 将下一个字节，即原寄存器 D0 中 8～15 位传输给外围设备
ROR.L   #8,D0         ; 重复
MOVE.B  D0,(2,A0)
ROR.L   #8,D0
MOVE.B  D0,(0,A0)
ROR.L   #8,D0         ; 4 次循环移位后回到 D0 原始的值
```

每传输一个字节，其拷贝的目标地址与当前地址间隔两个字节。每次数据传输后 ROR.L #8, D0 指令都将 D0 中的 32 位值右移 8 位使下次需要传输的字节位于最低的字节。而使用 MOVEP 指令，整个代码可以减少为：

```
MOVE.L  #Peri,A0      ; A0 指向存储器映射的外围设备
MOVEP.L D0,(A0)       ; 将 D0 中的字传输至外围设备
```

注意这段代码十分紧凑。MOVEP 是一条很好的、可以执行有用任务的指令，但其并不是一个必要的指令。

1. 保存 I/O 操作的顺序

RISC 体系结构只提供存储器加载和存储操作，并没有实现便于 I/O 操作的指令。然而，在有些情况下，RISC 组织和存储器映射的 I/O 会冲突。前面介绍过，一些存储器映射的外围设备具有配置和自动复位状态寄存器或者自动递增指针。以恰当的、程序员定义的序列来访问这些外围设备十分重要。因为超标量 RISC 处理器采用了机会主义方法来访问存储器，数据可以以乱序方式存储至存储器。这种乱序存储器访问不会对数据存储和检索造成问题，但它会破坏存储器映射的 I/O。

PowerPC 实现了 eieio（enforce in-order execution of I/O，强制顺序执行的 I/O）指令，该指令没有参数，但确保完成了所有之前启动的存储器访问。考虑以下这个例子，两个载入操作后面紧接着一个加法运算。

```
lwz  r5,1000(r0)      ; 将存储器地址为 1000 的内容加载至 r5
lwz  r6,1040(r0)      ; 将存储器地址为 1040 的内容加载至 r6
add  r7,r5,r6         ; r7=r5+r6
```

当执行这些指令时，处理器可以交换 r5 和 r6 从存储器加载的顺序。只要在加法运算前完成了两个加载操作，加法的结果就不依赖于加载的顺序。假设地址 1000 和 1040 是存储器映射的位置。例如，设计的外围设备在对地址 1000 进行读访问后修改了地址 1040 处的寄存器，两个加载指令的顺序就变得十分重要了，交换它们的顺序可能导致不正确的结果。

考虑下面这个需要更新外围设备的例子。因为寄存器需要通过指针来访问，在向指针寄存器所指示的寄存器中写入数据之前，需要先将寄存器地址写入指针寄存器。本例中，需要

向编号为35的外围寄存器加载数值99。PowerPC的代码为：

```
addi   r5,r0,35          ; r5=35（注意在PowerPC中r0总是0）
addi   r6,r0,99          ; r6=99
stw    r5,1234(r0)       ; 将35存储在1234这个位置（指针内容为1234）
stw    r6,5678(r0)       ; 将99存储在5678这个位置
```

这两个写操作必须以正确的顺序执行。为了确保这一点，PowerPC有3个同步指令：eieio、sync和isync。isync指令强制指令或存储事务在继续往下执行之前完成；也就是说，isync之前的指令被执行完，预取的指令被丢弃。然后，才开始新的取指。指令eieio在后续写操作之前强制所有未完成的写操作被完成。sync指令强制在执行后面的指令之前将所有以前的读和写操作在总线上完成。通过在写操作之间插入eieio指令，可以确保前面的代码以正确的顺序运行。

```
addi    r5,r0,35        ; r5=35
addi    r6,r0,99        ; r6=99
stw     r5,1234(r0)     ; M[1234]=35; 即将改变寄存器r5
eieio                   ; 在继续之前确保r5的内容被写入
stw     r6,5678(r0)     ; M[5678]=99; 寄存器值为99
```

2. 副作用

在结束介绍存储器映射的I/O之前，必须介绍指令副作用（side effect）这一概念。一条指令应该精确地完成它应该做的任务，不能有更多的功能。考虑68000处理器的CLR <ea>指令，它将清除指定有效地址（ea）处的操作数；也就是说，[ea]←0。很简单，不是吗？然而，CLR操作的内部实现为：

```
[Temp]  ← [ea]    ; 假读（CLR指令的副作用）
[ea]    ← 0       ; 实际的CLR操作
```

第一个操作为假读（dummy read，或哑读），其后为存储器清除操作。前文介绍过，有些外围设备需要使用读和写访问来区分寄存器对。CLR指令将导致对一个寄存器的假读，然后向另一个寄存器写入零。这是个问题吗？正确的寄存器将被清除，另一个具有相同地址的寄存器会被执行读操作。虽然对存储器的读操作是无害的，但是读取具有自动清除（self-clearing）标志位的状态寄存器会导致状态被清除。这里的CLR指令看上去很无辜，但可能会在存储器映射的系统中导致无法解释的副作用。

4.2 数据传输

要理解数据传输需要3个至关重要的概念：开环传输（open-loop transfer）、闭环传输（closed-loop transfer）和数据缓冲（data buffering）。开环传输过程中，信息发出后即被假定能够正确地接收。闭环传输过程中，接收方主动通知发送方数据已经接收。数据缓冲涉及处理数据传输速率和接收方接收（consume）速率之间的差异。

4.2.1 开环数据传输

传输数据最简单的方法就是将数据放在总线上并发出信号（或者数据选通）来表明数据已经有效。图4-11展示了在外围接口部件和外围设备（如打印机）之间的开环传输。处理器通过其地址和数据总线将数据移动到外围接口，外围接口再将数据放到总线上。

外围接口发出数据有效（data avaliable）选通信号$\overline{\text{DAN}}$向外围设备指示在其输入端的数据是有效的。外围设备读取数据，外围接口将作废$\overline{\text{DAN}}$来完成传输。

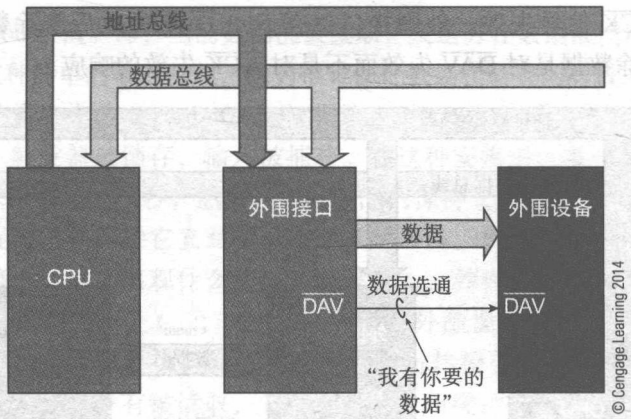


图 4-11 开环数据传输

图 4-12 给出了这一信息交换过程的时序图，它被称为开环是因为没有反馈来确认数据已经真正被接收。如果外围设备处于离线、忙碌或是速度很慢，数据有效（即 $\overline{\text{DAV}}$ 有效）的时间段内数据可能还没有被读取。开环数据传输也被称为同步（synchronous）传输，因为接收数据的设备必须与发送数据的设备同步。

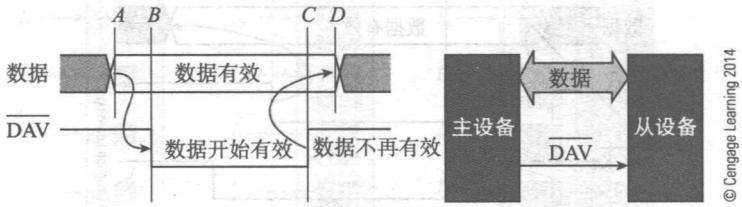


图 4-12 开环数据传输的时序图

4.2.2 闭环数据传输

在闭环传输过程中，接收数据的设备向发送方返回一个确认信号形成闭环。图 4-13 与图 4-11 类似，除了有一个以数据传输确认（data transfer acknowledge）信号 $\overline{\text{ACK}}$ 为标记的反馈路径。图 4-14 提供了相应的时序图。外围接口使数据有效，在 *B* 点发出 $\overline{\text{DAV}}$ 信号表明数据有效（与开环数据传输一样）。接收数据的外围设备发现 $\overline{\text{DAV}}$ 有效并读取数据。反过来，外围设备发出 $\overline{\text{ACK}}$ 通知接口数据已被接收。接口撤回 $\overline{\text{DAV}}$ 信号（使信号翻转失效）来完成数据交换。这种事件的序列被称为握手（handshaking）。握手通过将传输挂起直到外围设备发出 $\overline{\text{ACK}}$ 表明其已经就绪这种方式，可以支持慢速的外围设备。

图 4-14 中的时序图称为握手，因为 $\overline{\text{ACK}}$ 的发出对应 $\overline{\text{DAN}}$ 的发出。闭环数据传输的优点是数据的发送者知道数据已被接收，不会因为远程外围设备没有读取数据而造成数据丢失。

图 4-14 中的握手闭环协议可以再进一步。在图 4-14 中，发出 $\overline{\text{DAV}}$ 后由外围设备发出 $\overline{\text{ACK}}$ 信号进行匹配。此时，假定数据已经收到，数据交换完成。图 4-15 显示了完全互锁的握手（fully in ter lock ed handshaker）协议，其中事件的顺序被更严格地定义，每个事件按顺序触发下一个事件。

在图 4-15 中的 *B* 点，发出 $\overline{\text{DAV}}$ 信号表示数据有效，在 *C* 点，发出 $\overline{\text{ACK}}$ 信号表示数据被收到。该序列继续下去，在 *D* 点使 $\overline{\text{DAV}}$ 失效。外围接口可以使 $\overline{\text{DAV}}$ 失效是因为 $\overline{\text{ACK}}$ 信号表明 $\overline{\text{DAV}}$ 已经被识别。$\overline{\text{DAV}}$ 失效可以告诉外围设备其确认信号已经被检测到。因此，外

围设备在 E 点使 $\overline{ACK}$ 信号失效。外围接口也在 F 点 $\overline{DAV}$ 失效后删除数据。F 点可能出现在 E 点前，因为删除数据是对 $\overline{DAV}$ 失效而不是对 $\overline{ACK}$ 失效的响应。

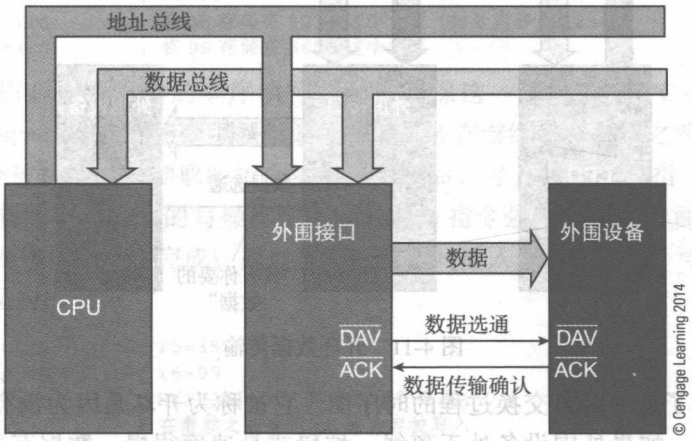


图 4-13 闭环数据传输

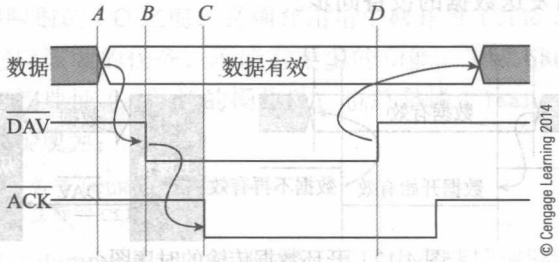


图 4-14 闭环数据传输的时序图

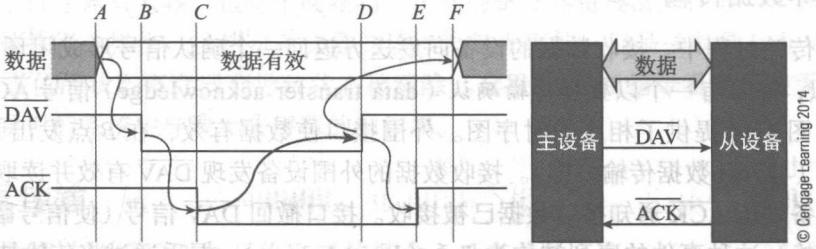


图 4-15 完全互锁的闭环数据传输的时序图

在所有包含握手的数据传输中，当发送方发出数据有效信号而接收方接下来没有发出数据确认信号时（例如，由于设备故障导致）就会出现问题。当发送方发出数据时，它在发出 $\overline{DAV}$ 信号时启动一个定时器。如果经过一段时间后接收方还没有发出确认信号，该操作就会被中止。从某个动作开始到声明为失败状态之间的时间称为超时（timeout）。当超时发生时，将产生中断，迫使计算机采取行动。

4.2.3 缓冲数据

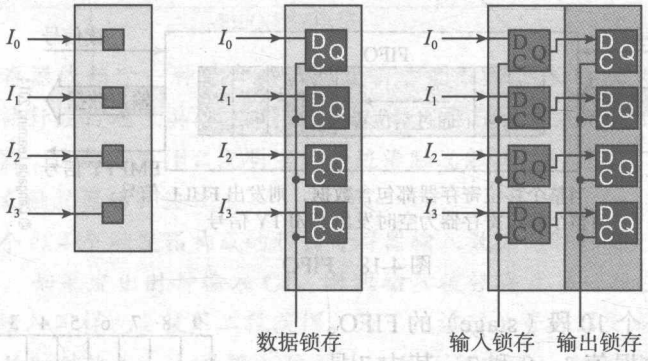
当数据在总线上传输时，可以在其有效时立即使用它，或者将其放入存储器设备。图 4-16 显示了外围设备中可能使用的 3 种输入电路。图 4-16a 的电路读取数据输入 $I_0 \sim I_3$

上的瞬时（instantaneous）值。即，当前数据值被读取，发送方在数据被使用时必须维持这些数据的值。很少实现这种简单的输入机制，因为很难保证数据在需要的时段内一直保持稳定。

图 4-16b 描述了单缓冲（single-buffered）或锁存（latched）输入，其输入连接到 D 触发器。当读取数据时，触发器被锁存，输入被捕获。在这种安排中，唯一的要求是，输入数据需要在其被锁定前保持有效 t_{setup} 秒，然后在后续时间内保持 t_{hold} 秒（这是锁存器的基本参数）。

单缓冲输入捕捉数据并保持它直到下一次锁存被启动。如果数据到达的速率与数据从输入锁存读取的速率接近，这会出现什么情况？假设每 t_{input} 秒到达新的输入数据，外围设备每 t_{cycle} 秒读取数据。如果 t_{cycle} 小于 t_{input}，一切 OK。假设外围设备在一段短时间内不能读取数据（例如，磁盘驱动器可能需要执行新的寻道或者执行热校正操作）。在这种情况下，输入数据在下一个数据到达之前没有被读取，它就会丢失。单缓冲输入在到达率可以超过读取速率时不能可靠地工作。

图 4-16c 为上一个数据被读取之前就会到来新数据的问题提供了一种解决方案。初始时，输入数据锁存如图 4-16b 一样。然而，输入锁存中的数据然后被复制到第二组锁存——输出锁存，数据在其中被第二次缓冲。这种安排意味着缓冲的输入端可以捕获数据而输出端在等待旧的数据被读取。当然，如果第三个数据元素接踵而至，这种安排就会失效。



a）不缓冲。它意味着从终端读取瞬时值

b）单缓冲。输入端的数据被锁存在 D 触发器中

c）双缓冲。数据首先被捕获至输入锁存，然后被拷贝至输出锁存

© Cengage Learning 2014

图 4-16 缓冲数据

图 4-17 给出了双缓冲输入系统的时序图。输入数据以固定的时间间隔到达。输入样本在时钟到来时每隔 C_{Ii} 时间被送至输入锁存，其中 i 为时钟脉冲号。

数据由时钟 C_{Oi} 锁定到输出锁存。在这种情况下，数据以不规则的间隔被读取。图 4-17 中有两个突出显示的例子。输入样本 i+3 早早地被送入输出锁存，但很久以后才被读取。同样，输入样本 i+5 读取得较晚而输入样本 i+6 读取得较早。正如读者所看到的，C_{O3} 和 C_{O4} 之间的时间间隔大于两个连续输入间的间隔，但由于采用了双缓冲技术，并没有出现数据丢失的情况。

FIFO

数据缓冲更一般的解决方案是提供先进先出（first-in-first-out，FIFO）的存储器。FIFO 是一种可以植入任意微机或外围设备内核的 n 级缓冲存储器。数据值被依次写入 FIFO 队列且按照相同的顺序读出。一旦数据被读取，就无法再次访问。FIFO 可以为空、部分填充或者充满；通常用输出标志来指示其为全空或者部分满。

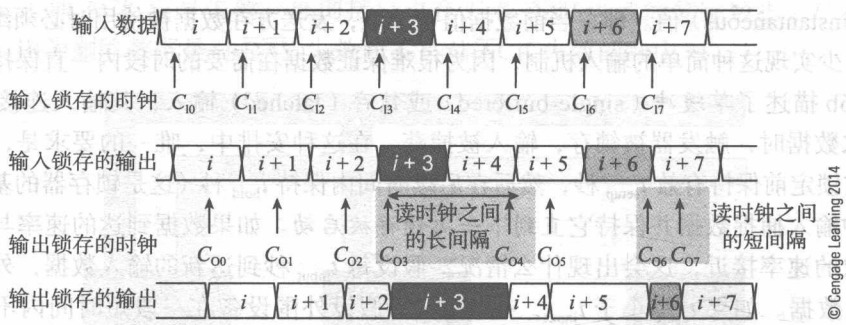


图 4-17 双缓冲输入的时序图

最简单的 FIFO 结构是一个具有输入端口（用于接收数据）和输出端口（用于提供数据）的寄存器。数据源提供了 FIFO 的输入和选通信号。同样，读取方提供选通信号表明其需要读取 FIFO 的数据。图 4-18 中的 FIFO 具有两个控制输出：FULL 表明 FIFO 不能再接收任何数据了，而 EMPTY 表明 FIFO 中没有任何可读数据了。可以把基于寄存器的 FIFO 理解为可以自动移位的移位寄存器。当数据到达输入端，它将在移位寄存器中行波（ripple）传送直到到达下一个空位置。

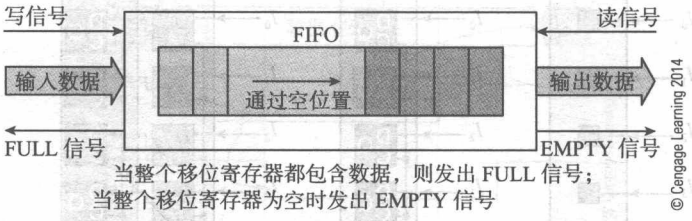


图 4-18 FIFO

图 4-19 展示了一个 10 段（stage）的 FIFO。初始时，FIFO 包含数据值 3、9 和 7，其中 7 是 FIFO 中最陈旧的数据元素，而 3 是最新的数据元素。在图 4-19b 中，8 被写入 FIFO，在图 4-19c 中，6 被写入。每个新的数据从输入端输入并排在队列的尾部。在图 4-19d 中，发生了读取数据操作，数据值 7 被从 FIFO 中移除。在这个段，所有数据元素向前移动一个位置。接下来是另一个写操作和读操作。下面将描述可作为一块单芯片的 FIFO。

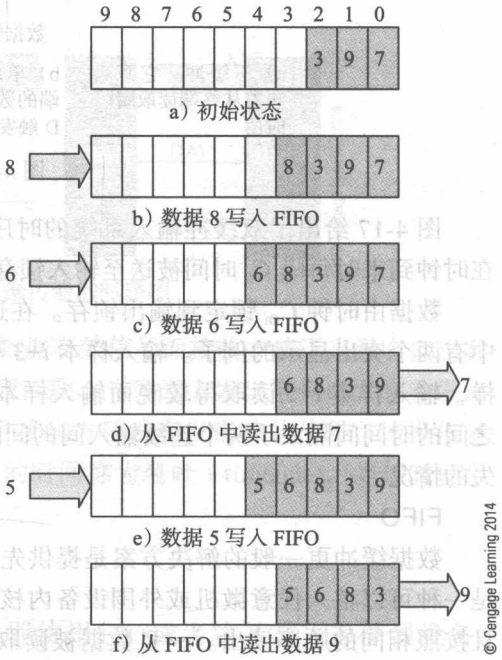


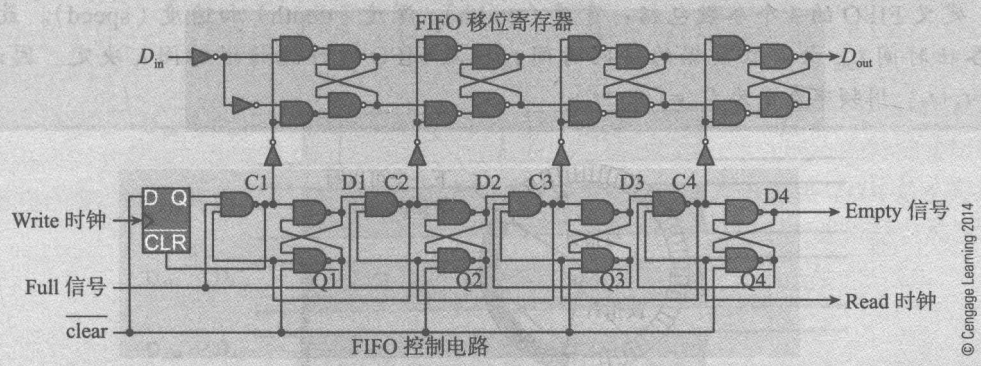
图 4-19 FIFO 中数据移动实例

FIFO 通常通过随机访问存储部件实现，这些存储部件被安排为循环缓冲区（如图 4-20 所示）。读指针和写指针用来指示 RAM 中的数据。图 4-21 说明了双端口 RAM FIFO 的结构。基于 RAM 的 FIFO 相比于基于寄存器的 FIFO 具有的优势是：基于 RAM 的 FIFO 中通过空位置的时间为常数，与其长度无关。该因素对于具有几千

个段的非常长的 FIFO 来说十分重要。

一种 FIFO 的实现

下图展示的 FIFO 使用移位寄存器 SN74SS225。这里只显示了 4 个 1 位的段。图的上部显示了由 4 个交叉耦合的与非门组成的 RS 触发器并通过 C1、C2、C3 和 C4 触发。当其中的一个段被触发时，来自前一个段的输入被锁定。因为所有段的输入都为 X 及其补码，RS 触发器的行为就像一个 D 触发器。



该移位寄存器依赖于一种通常被认为是有害的效应。当 D 或 RS 触发器被触发时，输入端的数据被行波传送至其输出端，如果在触发器将被触发时数据发生了变化，该数据将穿过触发器。这就是为什么发明主从和边沿触发触发器的原因。然而，FIFO 充分利用了电平触发式触发器的行波效应。

假设向一个由 4 个触发器构成的移位寄存器输入数据，时钟 C1 和 C4 为低电平。什么也没有发生。如果发出时钟输入 C1，数据输入被锁存至第一段。如果发出时钟 C2，第一段获得的输入同样也会被第二段获得。为了把该移位寄存器变成 FIFO，所要做的就是控制 4 个段的时钟输入。如果输入位置为 1 且队列中下一个空闲位置为 i，所要做的就是触发段 1 ～ i 的时钟，导致数据被行波传送至 FIFO。

电路的下部为 FIFO 的控制部分，有 5 个信号：Clear 信号对系统进行初始化，使得 FIFO 变成空状态；Write 时钟输入信号将数据输入 FIFO；Read 时钟输入信号将 FIFO 中最陈旧的数据项取出；Empty 信号的输出表明没有可以读取的数据；Full 信号的输出表明 FIFO 中所有段当前都被占用。

控制部分本身也像一个移位寄存器来进行安排，其输出为 FIFO 使用的时钟信号。如果控制移位寄存器的输出 Q 的前 i 个段被置为 1 时，这 i 个时钟均为高电平，数据将被行波传送至 FIFO 的第 i 段。这同时还提供了用来决定下一个数据元素放在 FIFO 哪个位置的标记。

控制移位寄存器的关键是它的反馈机制。每段的输出被反馈给左边的段。考虑 FIFO 非空时的输出段。D4 的状态为 1（即非空），时钟 C4 为 1（即 FIFO 输出还没有被时钟触发，数据仍然保存在输出端）。Q4 的输出是 D4 的补码，此时为 0。

当 Read 时钟为低时，Q4 的输出将被迫为高。然而，由于 Q4 为高，三输入的与非门的输出在控制状态下将变为高，此时其 3 个输入之一为高。同样，该与非门的输出将

向前反馈，控制段的行为与该段的写时钟类似。因此，将读时钟应用于控制移位寄存器的右侧将使其在控制电路中从右至左行波传送。然而，当到达某个被占用的段时，该段的输出为高，行波效应就会停止。

基于寄存器的FIFO的缺点是其硬件的复杂性、不灵活性及其不同的行波传输时间。当n段的FIFO为空，数据从输入段行波传送至输出段需要$n \cdot t_{stage}$秒，其中t_{stage}是每段行波传送的延迟。在深度较小时，基于寄存器的FIFO提供了具有成本效益的解决方案。当深度较深时，需要其他可替换的结构。

定义FIFO的3个参数包括：宽度（width）、深度（depth）和速度（speed）。最小的吞吐时间t_{min}由读取数据的访问时间t_A以及FIFO的行波传送时间t_F决定。因此，$t_{min}=t_A+t_F$。用频率表示为$f_{max}=1/(t_A+t_F)$。

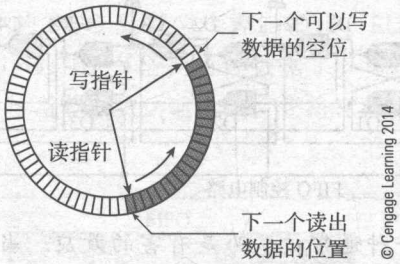


图4-20 基于存储器的FIFO的逻辑安排

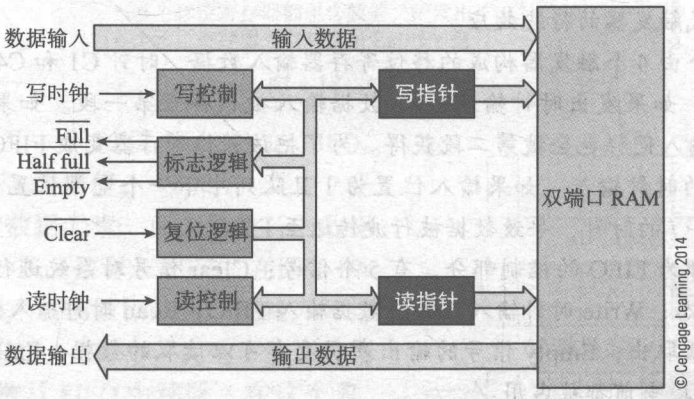


图4-21 基于存储部件的FIFO结构

半导体制造商制造了两种基于RAM的FIFO：异步（asynchronous）和FIFO同步（synchronous）FIFO。当其移入和移出信号来源独立，FIFO的操作为异步的；当移入和移出信号来自相同的时钟，它们之间存在精确的关系，则FIFO在同步模式下操作。相比异步FIFO，人们首选同步FIFO。

除了缓冲I/O事务外，FIFO还可以提供更多功能。图4-22演示了Texas Instruments公司的FIFO在某系统中的使用，该系统具有使用小端I/O的32位计算机和使用大端I/O的8位端口。该FIFO可由用户配置，可以设置为执行总线匹配（bus matching）；也就是说，其输入和输出总线可能有不同的宽度。这里，其端口A的接口为32位宽，其端口B的接口为8位宽。此外，还可以对其编程实现当数据从小端系统向大端系统拷贝时的数据交换。图

4-23 给出了将两个 32 位的数据字写入 FIFO 以及从 FIFO 中读出 8 个字节的时序图。

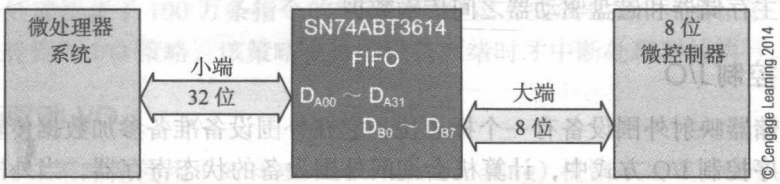


图 4-22 使用 FIFO 将系统与不同的总线宽度和端格式相连

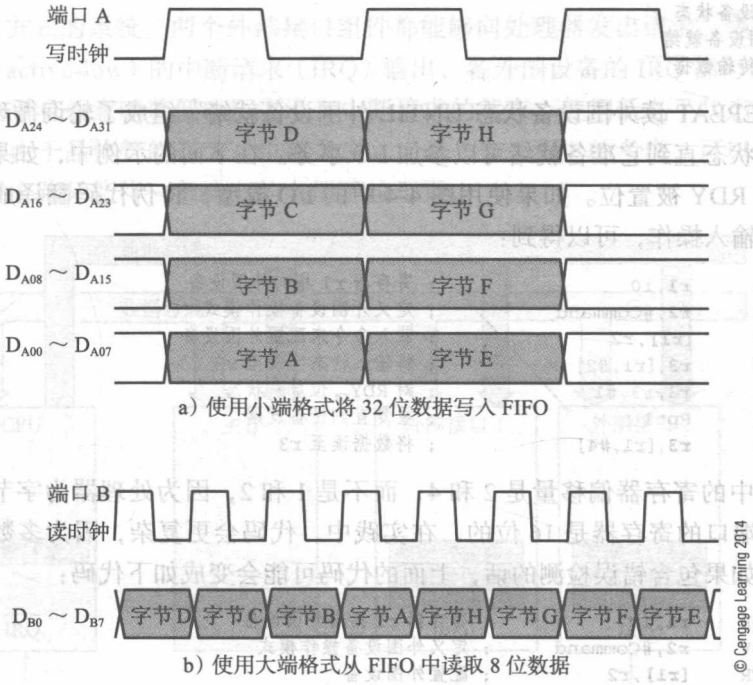


图 4-23 图 4-22 对应的时序图

至此，本书已经介绍了将数据移入和移出计算机的词汇和基本策略。后文将对 I/O 性能进行概述。

4.3 I/O 策略

计算机通过 3 种方式实现 I/O 事务处理。它可以使用*程序控制* I/O（programmed I/O）方式处理单独的 I/O 事务。它可以使用*中断驱动* I/O（interrupt-driven I/O）方式在外围设备发出参与 I/O 事务就绪信号之前执行其他任务。它可以使用*直接存储器访问*（direct memory access，DMA）方式通过专用硬件来执行 I/O 事务处理。计算机系统还可以使用这些策略的混合策略。本节首先介绍简单的程序控制 I/O 机制，通过执行适当的输入或输出指令将数据移入或移出系统。本节的主要部分介绍外围设备用于请求数据传输的中断机制。中断驱动的 I/O 比程序控制 I/O 更有效，因为它只在外围设备就绪时发生。第三种 I/O 策略，DMA，是最复杂的一种策略，因为它需要一个控制子系统接管总线控制和在外设和存储器之间初始化数据传输。处理器可以根据实际配置在 DMA 操作期间暂停或者可以将其自己的操作与

DMA 周期交织在一起。例如，当 CPU 从它自己的内部缓存读取指令和数据时，DMA 控制器可以直接在主存储器和磁盘驱动器之间传输数据。

4.3.1 程序控制 I/O

典型的存储器映射外围设备有一个标志位，它在外围设备准备参加数据传输时被外围设备置位。在程序控制 I/O 方式中，计算机会询问外围设备的状态寄存器，当外围设备准备就绪后进行处理。可把这一操作用伪代码描述为：

```
REPEAT
    读外围设备状态
UNTIL 外围设备就绪
与外设之间传输数据
```

操作“REPEAT 读外围设备状态 UNTIL 外围设备就绪”组成了轮询循环，需要不断测试外围设备的状态直到它准备就绪可以参加 I/O 事务。在下面的示例中，如果外围设备有数据，则状态位 RDY 被置位。如果使用图 4-4 中的 I/O 模型，将伪代码翻译成一般的汇编语言形式来执行输入操作，可以得到：

```
        ADR     r1,io              ; 寄存器 r1 指向外围设备
        MOV     r2,#Command        ; 定义外围设备操作模式
        STR     [r1],r2            ; 载入命令来配置外围设备
Rpt1    LDR     r3,[r1,#2]         ; 将输入状态字读至 r3
        AND     r3,r3,#1           ; 对 RDY_IN 位屏蔽状态
        BEQ     Rpt1               ; 重复直到设备就绪
        LDR     r3,[r1,#4]         ; 将数据读至 r3
```

外围设备中的寄存器偏移量是 2 和 4，而不是 1 和 2，因为处理器为字节寻址设备，而这里假定 I/O 端口的寄存器是 16 位的。在实践中，代码会更复杂，因为多数状态寄存器都包含错误位。如果包含错误检测的话，上面的代码可能会变成如下代码：

```
        ADR     r1,io              ; 寄存器 r1 指向外围设备
        MOV     r2,#Command        ; 定义外围设备操作模式
        STR     [r1],r2            ; 配置外围设备
Rpt1    LDR     r3,[r1,#2]         ; 将输入状态字读至 r3
        MOV     r4,r3              ; 将状态字的副本拷贝至 r4
        AND     r4,r4,#Error       ; 向全局错误位屏蔽状态
        BEQ     BigFault           ; 处理错误
        MOV     r4,r3              ; 为下次测试恢复状态
        AND     r4,r4,#RDY         ; 对 RDY_IN 位屏蔽状态
        BNE     Rpt1               ; 重复直到设备就绪
        MOV     r4,r3              ; 为下次测试恢复状态
        AND     r4,r4,#OK          ; 寻找操作错误
        BEQ     TinyError          ; 处理错误
        LDR     r3,[r1,#4]         ; 读取数据
```

输入子程序测试 Error 状态位，该位在端口不能正常工作时被置位。接下来，检查就绪状态位。如果就绪位被置位标识新的输入，将检查另一个错误位——OK，来测试当前输入是否出现错误。

如果状态位是自动清除的，必须将状态字保存至寄存器然后在需要它的时候再从寄存器中拷贝。每次需要测试状态字中的一位时，就可以从 r3 中重新获得该状态字。

程序控制的 I/O 由于低效并没有被广泛采用。假设平均每条指令需要 t_{inst} 秒执行时间，I/O 事务每 $T_{I/O}$ 秒发生一次。在 I/O 事务中，处理器处于轮询循环中，而处理器本可以执行 $T_{I/O}/t_{inst}$ 条有用的指令。例如，如果处理器速度为 10 000 000 条指令 /s，t_{inst}=100ns，I/O 设备

为键盘，其操作速度为 $1/T_{I/O}$=10 个字符 /s，$T_{I/O}/t_{inst}$ 的结果为 1 000 000。也就是说，对于每个输入操作，处理器花了 100 万条指令的执行时间但没有完成有用的工作。在下一节中，将介绍可以避免轮询的 I/O 策略，该策略在外围设备就绪时才中断处理器的执行。

4.3.2 中断驱动 I/O

更高效的 I/O 策略是使用中断处理（interrupt handling）机制来处理 I/O 事务的发生。即处理器执行其他的任务，直到外围设备发出需要处理的请求。当外围设备就绪时，它中断（interrupt）处理器，进行事务处理，然后处理器返回中断前的状态。图 4-24 描述了一个使用中断驱动 I/O 方式的系统。两个外部接口组件都能够向处理器发出请求。多数外围设备具有低电平有效（active-low）的中断请求（$\overline{IRQ}$）输出，各外围设备的 $\overline{IRQ}$ 都与处理器 $\overline{IRQ}$ 的输入连接。低电平有效意味着低电压表示有中断请求的状态。使用低电平状态来表示有效状态的原因完全是由于晶体管的行为；也就是说，它是一个工程上的考虑，可以追溯到集电极开路的时代，该电路只能将一条线上的电压下拉至零。

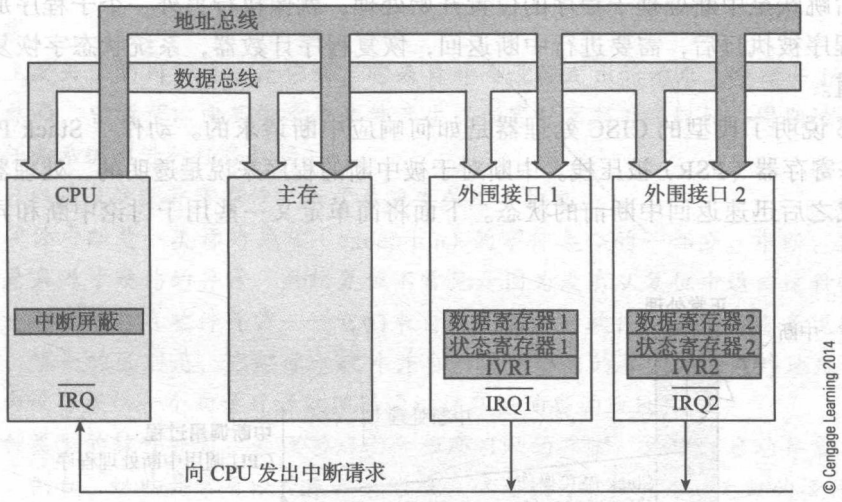


图 4-24 中断驱动 I/O 的基本结构

每当外围设备想参加 I/O 事务，它发出 $\overline{IRQ}$ 信号，并使 CPU 的 $\overline{IRQ}$ 输入为低电平有效。CPU 检测到其 $\overline{IRQ}$ 已经有信号，如果该请求没有被屏蔽，则响应该中断请求。大多数处理器都具有中断屏蔽寄存器（interrupt mask register），如果 CPU 在执行重要操作时允许关闭中断。即当处理器在执行关键任务时可以屏蔽中断。例如，使用实时监测快速事件的系统不会响应键盘输入的中断（即使是最快速的电脑打字员相对计算机的内部操作来说也是极其缓慢的）。同样，从系统故障（例如掉电）中恢复也具有这种屏蔽中断的优先权。

处理器响应中断的方式与设备相关。图 4-24 中的两个外围设备连接到相同的 $\overline{IRQ}$ 信号线，CPU 不能确定是由哪个设备发出了中断请求。CPU 通过查询（polling）每个外围设备的状态寄存器来定位中断源。中断查询机制提供了中断优先级（interrupt prioritization），因为重要设备的中断请求需要被迅速响应而被首先查询。

在图 4-24 中，每个存储器映射的外围设备具有中断向量寄存器（interrupt vector register，IVR），告诉处理器如何找到合适的中断处理程序。通常，IVR 提供中断向量表的

指针。

1. 中断处理

中断是异步（asynchronous）事件，因为处理器不可能知道外围设备（如键盘）何时将产生中断。当中断发生时，计算机首先决定是响应它还是忽略它。从 CPU 接收到中断请求开始到响应中断的时间被称为中断延迟（interrupt latency）。计算机响应中断时，它执行以下操作顺序。

- 完成当前指令。指令是不可分割的（indivisible），必须被完整地执行。
- 程序计数器（program counter，PC）的内容被保存以允许程序可以从断点处继续执行。CISC 处理器将程序计数器保存在栈中，故中断自己还可以被中断，而不会丢失其返回地址。大多数 RISC 处理器将 PC 保存在链接寄存器（link register）中。
- 处理器的状态也必须保存。处理器的状态由条件码的标志位以及其他状态信息来定义。假设某条指令设置了 Z 位，下一条指令需要测试 Z 位。显然，如果在设置 Z 位和测试 Z 位的指令之间发生了中断，中断机制必须保证 Z 位的状态不被修改。
- 然后跳跃至中断处理子程序的位置开始处理，就像执行另外一个子程序那样。在该子程序被执行后，需要进行中断返回，恢复程序计数器，系统状态字恢复至中断前的值。

图 4-25 说明了典型的 CISC 处理器是如何响应中断请求的。动作“Stack PSR”表示处理器状态寄存器（PSR）被压栈。中断对于被中断的程序来说是透明的，处理器必须在中断处理完成之后迅速返回中断前的状态。下面将简单定义一些用于讨论中断和异常的关键概念。

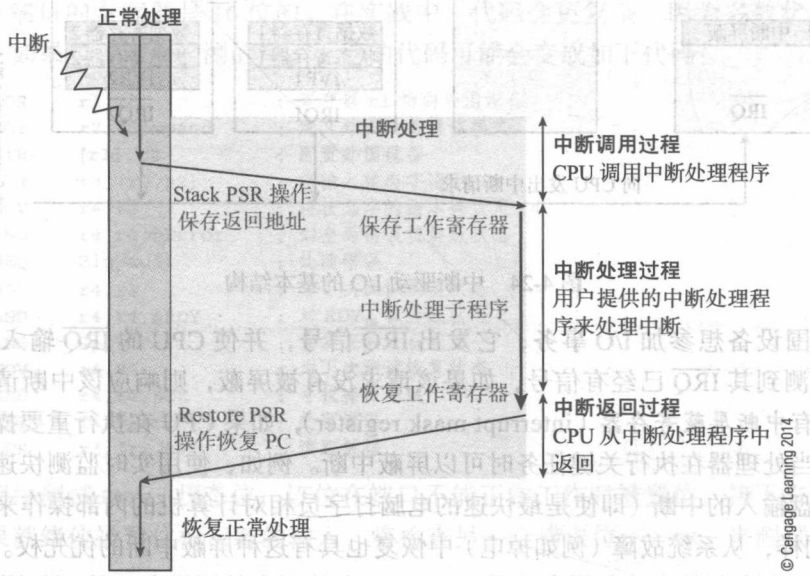


图 4-25 使用栈来保存返回地址的中断处理过程

2. 不可屏蔽中断

有一类中断请求（interrupt request）被称为不可屏蔽（nonmaskable）中断，这是因为其不可以被拒绝或被延迟（deferred）。有时需要计算机必须响应中断，无论它正在做什么。某些微处理器具有不可屏蔽的中断请求（NMI），该请求不能被延迟必须被服务。不可屏蔽中

断必须被保留来处理类似掉电这样的事件。在此情况下，低压检测器在电力开始下降时尽可能快地产生不可屏蔽中断。NMI 处理子程序使处理器处理中断，在系统电力下降至使计算机完全失效的危险水平之前有序地关闭系统。

在多个设备可以发出中断请求的环境中，需要一种机制来区分重要和次重要的中断。例如，如果由于磁盘具有可以被处理器读取的数据而使控制器产生中断，中断必须在数据可能被磁盘驱动器中的新数据替换而失效之前被处理。另一方面，由键盘接口产生的中断在其被服务之前可能有 200ms 至几秒的时间，因此，从键盘来的请求在其他设备发出需要立即服务的请求时可能被迫等待。

3. 中断优先级

微处理器通常支持带有优先级的中断（即芯片具有多个中断请求输入）。每个中断都有预定义的优先级，具有与当前中断相同或更低优先级的新中断不能中断处理器，直到当前中断已被处理。同样，具有更高优先级的中断可以中断当前中断的处理。稍后将介绍 IA32 的中断处理机制。

中断和异常

本章主要关注的内容之一是中断，它来自外围设备发出的请求。中断与 I/O 操作密切相关。然而，中断可以由其他硬件事件产生，如定时器超时（例如，周期性的中断可用于在多任务系统中进行任务切换）。

中断可以用来处理异常事件，如子系统失效或者掉电。

这种硬件中断是一类称为异常（exception）的事件集合的一部分。中断、复位以及页故障都是来源于硬件的异常，虽然复位不常见是因为没有从复位中返回这种情况。与硬件异常一样，还存在软件异常——它们来自处理器中的软件。软件异常表现得完全像硬件异常，唯一的区别是，它起源于软件并自动提供恰当的异常处理程序地址（不像硬件中断，由设备提供一个向量或者处理器通过循环轮询查询地址）。

有两种类型的软件异常：处理器启动和程序启动的异常。处理器启动异常总是由于错误导致。例如，这些异常可以是零作为除数、试图执行非法指令（无效的操作码）、特权违规（用户试图执行保留给操作系统的操作）或者未对齐的访问（访问 32 位字的奇数字节地址）。

用户产生的软件异常包括系统调用和仿真器自陷。在系统调用中，一条称为管理程序调用（supervisor call）或陷阱（trap）的特殊指令被插入代码中，调用是由操作系统执行的函数，如控制台 I/O 事务或磁盘读/写。陷阱和子程序的区别是，陷阱并不需要子程序那样的显式的目标地址，因为陷阱的目标被嵌入处理器的硬件中。此外，陷阱是可移植的，它可以用在所有具有相同操作系统的系统中。

仿真器陷阱提供了执行尚未被设计（或者运行代码的实际芯片不具备）的指令的一种手段。例如，低成本的处理器可能没有硬件浮点运算单元。仿真器陷阱可以用来代替浮点指令。当处理器遇到仿真指令，操作系统被调用，该指令通过软件进行仿真。这种机制可用于确保处理器系列的软件兼容。

4. 中断嵌套

中断和其他处理器异常具有子程序的所有特点，其返回地址在调用开始时被压栈，一旦

子程序已经执行完成时返回地址被恢复。中断就是一种子程序调用，它由硬件或者软件自动提供目标地址，并且提供了保护条件码状态以及程序计数器的机制。

正如子程序可以嵌套那样，中断也可以嵌套。图 4-26 显示了 1 级中断被调用、处理、然后返回正常处理的过程。接着 1 级中断第二次发生。然而，在此情况下，1 级中断处理程序完成任务之前发生了 2 级中断。此时，1 级中断处理程序被中断（暂停），开始进行 2 级中断处理。当 2 级中断已被处理，返回 1 级中断处理程序，完成被中断（暂停）的中断。

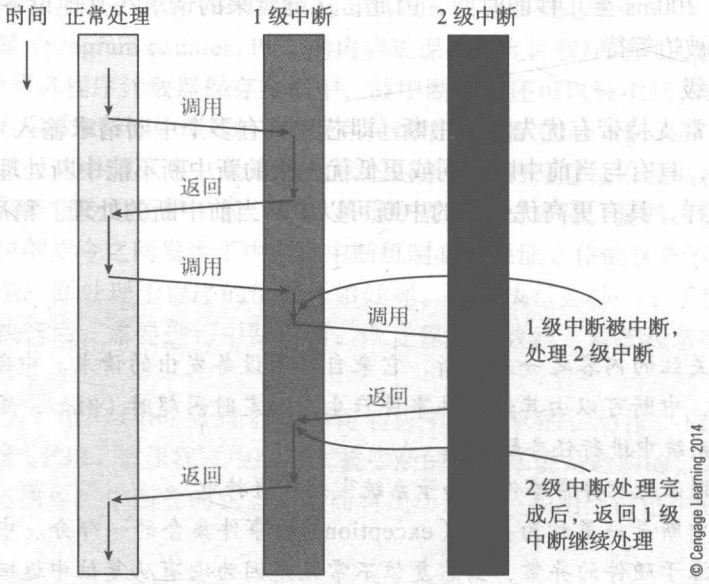


图 4-26 中断嵌套

考虑下面这个中断优先级和中断嵌套的例子。系统有 6 个级别的中断优先级，每隔 10μs 发生一个中断。假定所有中断需要 15μs 才能完成。表 4-2 按照中断的优先级给出了它们的顺序，图 4-27 展示了它们（在这个假想的例子中）是如何被执行的。

表 4-2 优先级中断实例

时间槽 /μs	中断级	时间槽 /μs	中断级	时间槽 /μs	中断级	时间槽 /μs	中断级
0	无	30	无	60	4	90	1
10	4	40	3	70	5	100	无
20	5	50	2	80	无	110	无

5. 中断向量

当具有单个中断请求线的处理器检测到服务的请求，它并不知道是哪个设备发出的请求，在它确定中断源之前，无法开始执行适当的中断处理程序。中断向量（vectored interrupt）通过要求发出请求的设备向处理器表明身份来解决中断源的识别问题。没有中断向量，处理器必须检查每个外围设备的中断状态位。

当处理器检测到中断请求时，它向所有可能发出中断请求的外围设备（中断源）广播（broadcast）中断确认（interrupt acknowledge）。每个可能的中断源检测来自 CPU 的确认信号，发出中断的设备返回一个向量（vector），CPU 用该向量调用相应的中断处理程序。图 4-28 展示了 68000 系列如何实现带优先级的（prioritized）和向量的（vectored）中断。图中有 7 个级别的中断请求，如果 $i > j$ 的话，第 i 级中断可以在第 j 级中断执行过程中被执行。

该机制允许中断嵌套，只有当 $j > i$ 的时候，第 i 级中断才可以被第 j 级中断所中断。

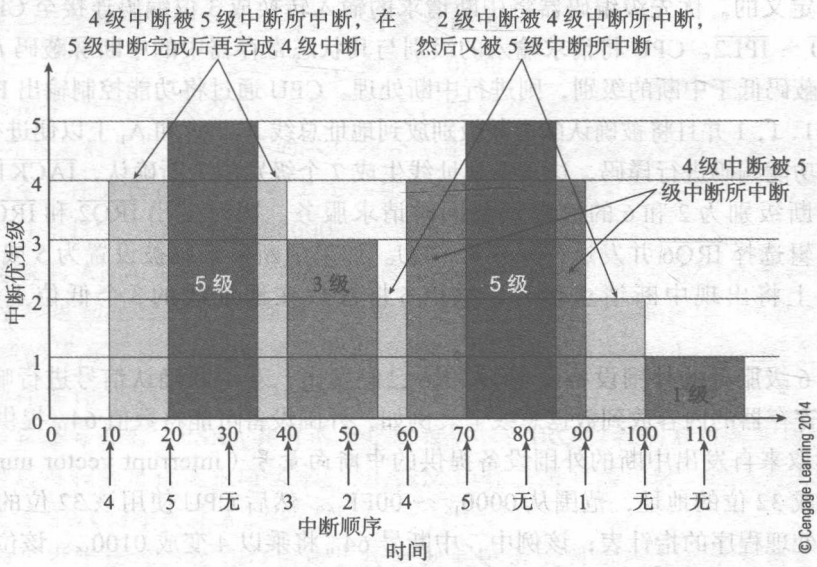


图 4-27 中断优先级实例

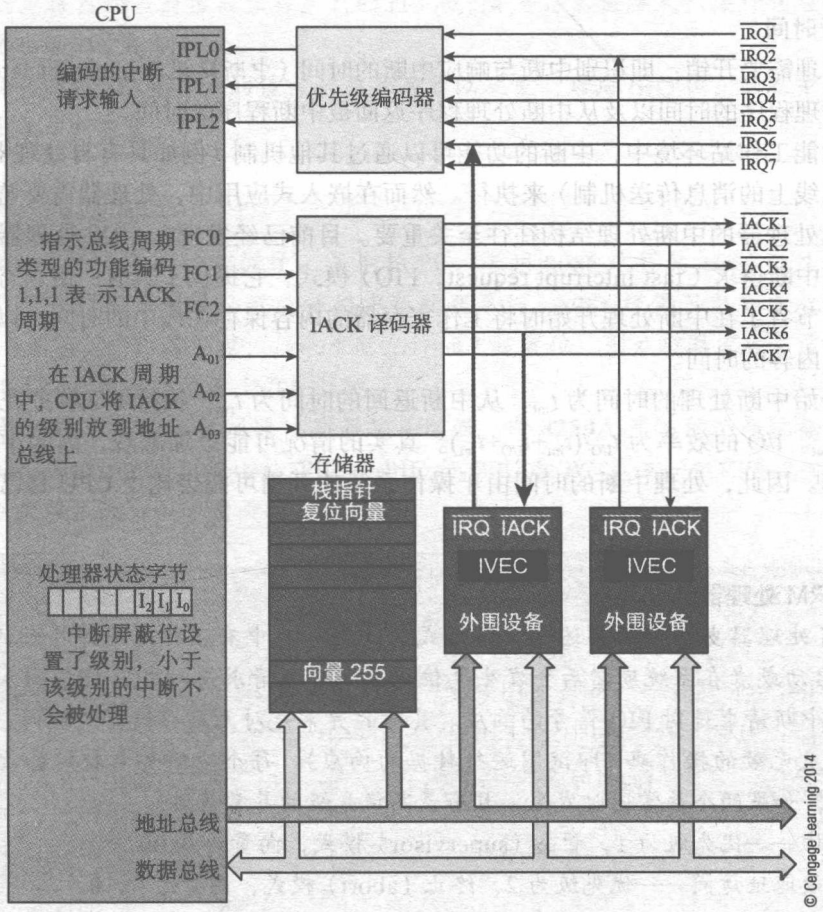


图 4-28 具有优先级和向量的中断机制

每个外围设备的$\overline{IRQ}$信号输出连接到7级中断请求$\overline{IRQ1}$～$\overline{IRQ7}$。外围设备产生的中断级是用户定义的。优先级编码器将中断请求的输入转换成3位编码连接至CPU中断优先级引脚$\overline{IPL0}$～$\overline{IPL2}$。CPU将请求输入的级别与其状态寄存器中的中断屏蔽码I_0～I_2比较。如果中断屏蔽码低于中断的级别，则进行中断处理。CPU通过将功能控制输出FC2，FC1和FC0设置为1，1，1并且将被确认的中断级别放到地址总线A_3、A_2和A_1上以便进行中断确认。外部逻辑对功能编码进行译码，这3条地址线生成7个级别的中断确认，$\overline{IACK1}$～$\overline{IACK7}$。

假设中断级别为2和6的外围设备同时请求服务。同时发出$\overline{IRQ2}$和$\overline{IRQ6}$信号。优先级编码逻辑选择$\overline{IRQ6}$并发送代码6至CPU。如果中断屏蔽码被设置为5或以下，FC2、FC1和FC0上将出现中断请求信号，数值6将被放在地址线的3个低位上，故发出了$\overline{IACK6}$信号。

请求第6级服务的外围设备发现$\overline{IACK6}$已经发出，对中断确认信号进行响应，将其8位中断向量寄存器的内容放到数据总线上。例如，外围设备可能将数值64_{10}提供给CPU。

CPU读取来自发出中断的外围设备提供的中断向量号（interrupt vector number），将其乘以4并形成32位的地址，范围从0000_{16}～$00FF_{16}$。然后CPU使用该32位的值来索引存储器中中断处理程序的指针表；该例中，中断号64_{10}将乘以4变成0100_{16}。该位置的内容代表该外围设备对应的中断处理程序的起始位置，它可以从存储器中读出然后加载到程序计数器中。

6. 中断时间

中断处理需要开销：即识别中断与响应中断的时间（中断延迟，interrupt latency）、调用对应中断处理程序的时间以及从中断处理程序返回被中断程序的时间。

在高性能工作站环境中，中断的功能可以通过其他机制（例如具有自处理器的专用I/O系统或者总线上的消息传送机制）来执行。然而在嵌入式应用中，处理器需要监视许多实时系统，因此处理器的中断处理结构往往至关重要。目前已经发现，ARM处理器系列实现了特殊的快速中断请求（fast interrupt request，FIQ）模式，它提供了具有一组新寄存器的中断处理程序，节省了在中断处理开始时将工作寄存器的内容保存在栈中的时间，以及在中断返回前恢复其内容的时间。

如果开始中断处理的时间为t_{int}，从中断返回的时间为t_{ret}，执行I/O事务花费的总时间为$t_{int}+t_{I/O}+t_{ret}$。I/O的效率为$t_{I/O}/(t_{int}+t_{I/O}+t_{ret})$。真实的情况可能更加悲观，因为中断由操作系统进行处理。因此，处理中断的时间由于操作系统的开销可能会比由CPU自己来处理中断要长得多。

中断和ARM处理器

ARM处理器支持下面描述的7种形式的异常。其中有3种是硬件导致的异常（中断）。在启动或者在系统崩溃后会发生复位。出现中断请求是对发出IRQ输入信号的响应，快速中断请求是对FIQ信号的响应。其他的异常是对无效存储器的访问、断点、软件中断以及无效的操作码（即试图运行数据的响应）。每个中断都与特定的优先级相关联，所以，如果两个异常同时发生，具有最高优先级的异常获胜。

1. 复位——优先级为1，管态（supervisor）模式，向量为0x00。
2. 无效地址访问——优先级为2，终止（abort）模式，向量为0x10。
3. FIQ——优先级为3，FIQ模式，向量为0x1C。

4. IRQ——优先级为 4，IRQ 模式，向量为 0x18。

5. 断点——优先级为 5，终止模式，向量为 0x0C。

6. SWI——优先级为 6，管态模式，向量为 0x08。

7. 非法指令——优先级为 6，未定义模式，向量为 0x04。

ARM 将相应的异常处理程序的入口指令（通常是分支指令或跳转到实际异常处理程序的跳转指令）存储在存储器中的上述地址。例如，复位异常处理的入口地址（将指针初始化指向引导程序）为 0×00000000。

前文曾谈到 ARM 是通过执行模式更改来响应异常的，在此过程中保存 r14 和 r13 中的状态字和返回地址。然后为当前异常创建新的影子（shadow）——寄存器 r14 和 r13。这些新的寄存器在 ARM 的术语中被称为分组寄存器（banked register）。快速中断请求分组寄存器 r8 ～ r12 以及 r13 和 r14。

处理器状态寄存器（PSR）具有两个中断控制位：IRQ 位控制启用和禁用中断请求；FIQ 位控制启用和禁用快速中断请求。当 ARM 工作在用户模式时，进入特权模式的唯一方法是通过 SWI 指令或异常。

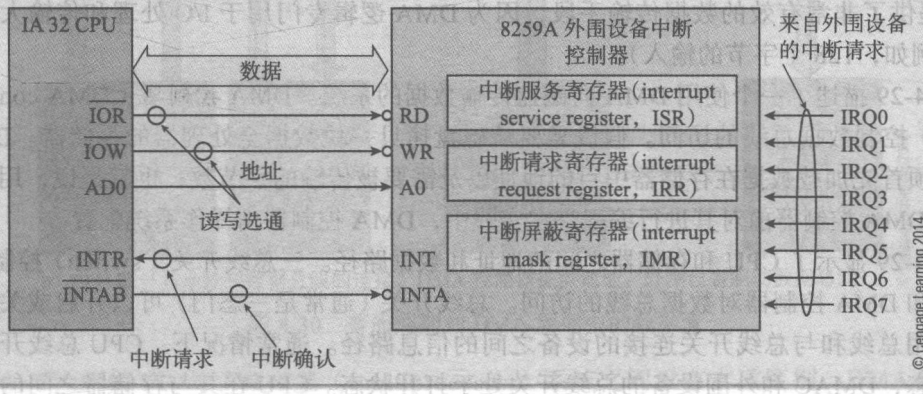
考虑如下中断请求。在 IRQ 引脚上检测到电平由低到高产生了中断请求。如果在 CSR 中的中断请求位被启用，IRQ 屏蔽位被禁用以阻止进一步的中断，中断才被处理。处理器状态寄存器的当前值被保存，新的 r13 和 r14 寄存器被换入。执行相应的中断处理程序，然后返回中断调用处。

如果保存当前 r13 和 r14 的值、状态寄存器以及重新启用 IRQ 位，则在中断处理程序中可以出现中断嵌套。

PC 环境下的中断处理

下面为 PC 中的中断处理概述。早期的 PC，采用 8086 这样的 16 位微处理器，支持比 68000 微处理器系列复杂度更低的中断处理系统。8086 提供了单个中断请求输入（INTR）。当发出 INTR 信号时，如果中断允许标志被置位，则处理器完成当前指令的执行，输出两个中断应答脉冲至 INTA 引脚。

8086 的原始中断处理机制通过外部芯片增强，即 8259A 可编程外部设备中断控制器（peripheral interrupt controller，PIC）。PIC 与 CPU 和 PC 系统一起实现带优先级和向量化的中断处理系统，如下图所示。



中断请求线 IRQ0 ～ IRQ7 连接到 PIC 的 8 位中断请求寄存器（IRR）。发出中断请求 IR_i 将使 IRR_i 置位。PIC 是可编程的，允许用户屏蔽任意的中断请求位。PIC 可以实现循环旋转（round-robin）的优先级，使得各种中断级可以公平地访问处理器，因为当前被服务的中断将排在下一轮仲裁的最后。

当中断请求正在等待处理时，INT 的输出（连接到 CPU 的 INTR 输入）发出请求信号。当 PIC 收到第一个来自 CPU 的 INTA 确认脉冲，它将 IRR 寄存器中最高优先级位清除，在中断服务寄存器（ISR）中将相应位置位。中断请求寄存器中，优先级最高的中断请求被从队列中删除，并传递到中断服务寄存器。剩下的中断等待处理，当前中断正在处理时可以接收另一个与之具有相同优先级的中断。来自 CPU 的第二个中断确认脉冲使 PIC 将 8 位的中断号放到总线上，CPU 使用其调用相应的中断处理程序。

ISR 的某位 ISR_i 必须被重置以表明当前中断已完成服务。用户可以选择两种方式来清除 ISR_i。在自动结束中断模式中，ISR 位是自动清除的，在第二个 INTA 脉冲结束时被重置。在手动模式下，CPU 必须在中断处理子程序结束时清除 ISR 位。注意 CPU 可以选择清除最高的服务位或者指定的服务位。如果实现了嵌套中断，最高级的服务位应该被清除，因为它对应最近被应答和服务的中断。

PIC 和 CPU 的组合完全支持中断嵌套。如果某中断请求的优先级比当前正在被服务的中断的优先级要低，它就会被忽略并保持挂起状态。但是如果发生了更高优先级的中断请求，该请求就会在当前中断完成之前被处理。中断不能被打断直到 INTA 序列已经完成。

8259A 可以级联，这意味着每个 PIC 的 8 个中断请求输入可连接到另一个 PIC 的 INT 输出；也就是说，可以将 8 个 PIC 与一个 PIC 相连，这样就可以允许 8×8=256 种优先级的中断。8259A PIC 是一种用于与旧的 PC 技术相结合的传统装置，如现在已经过时的 ISA 总线（PIC 控制 ISA 总线上来自设备的中断请求）。今天的 PC 机使用复杂的总线控制器子系统进行中断处理。

4.3.3 直接存储器访问

I/O 处理最复杂的方式是采用直接存储器访问（direct memory access，DMA），此时外围设备和存储器之间的数据传输不需要处理器的干预。实际上，DMA 使用了专用的处理器来执行 I/O 事务，它通过控制系统总线并利用总线完成外围设备和存储器之间的数据移动。DMA 提供了非常有效的数据传输手段，因为 DMA 逻辑专门用于 I/O 处理和传输大量突发数据（例如，128 个字节的输入）。

图 4-29 描述了一个使用 DMA 向磁盘传输数据的系统。DMA 控制器（DMA controller，DMAC）控制数据总线的访问。假设需要从磁盘拷贝一块数据至处理器的存储器。DMA 控制器必须首先加载数据在存储器中目的地址以及需要被传输的字节数；也就是说，用户需要在启动 DMA 控制器前对其进行编程。在实践中，DMA 控制器由操作系统配置。

图 4-29 显示了 CPU 和存储器之间的地址和数据路径。三总线开关（switch）控制 CPU、存储器和 DMA 控制器对数据总线的访问。总线开关（通常是三态门）可以开启或关闭以启用或禁用总线和与总线开关连接的设备之间的信息路径。通常情况下，CPU 总线开关处于关闭状态，DMAC 和外围设备的总线开关处于打开状态。CPU 在其与存储器之间的数据传

输通过将地址放到地址总线然后再读写数据来完成。图 4-30a 说明了 CPU 控制总线的情况，图 4-30b 显示了 DMA 控制器是如何控制数据总线并进行数据传输的。

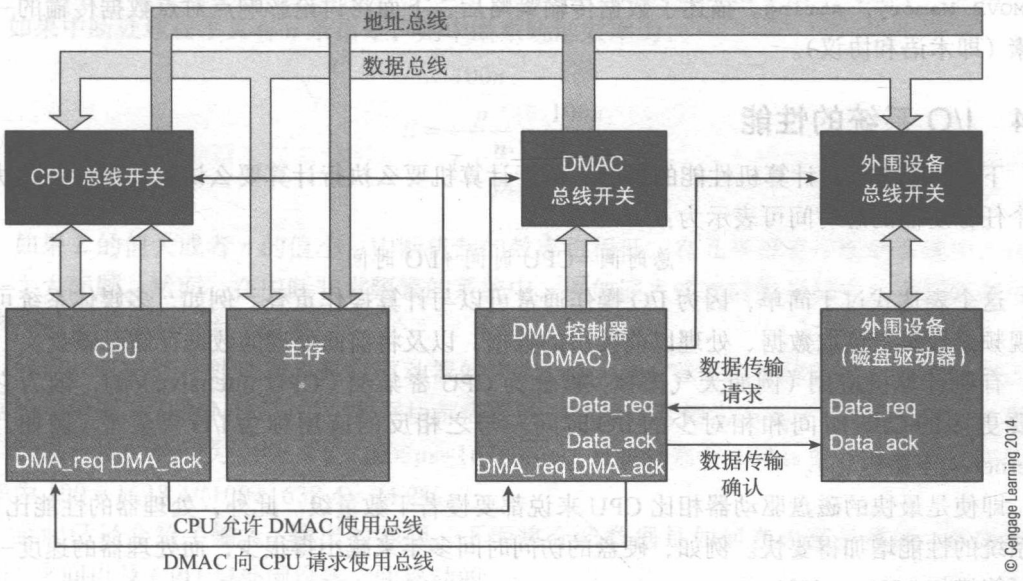
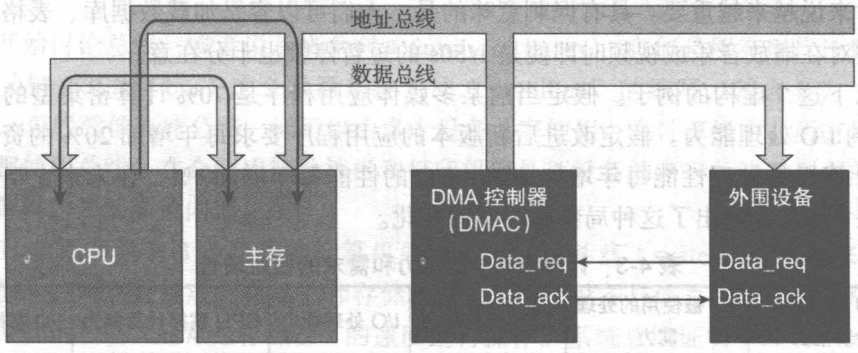
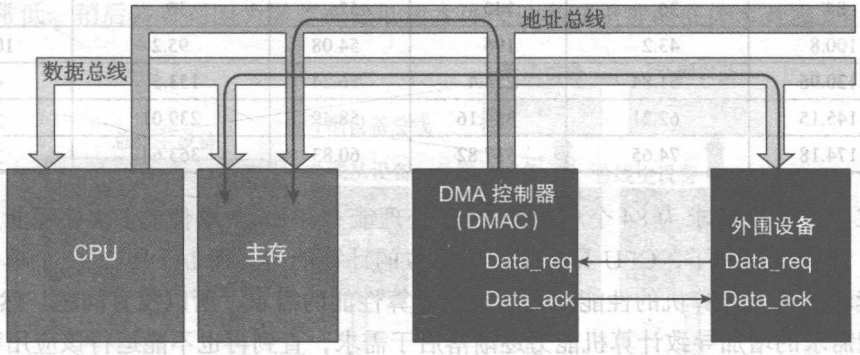


图 4-29 DMA 逻辑概览



a) CPU 访问存储器。DMA 控制器与地址总线不相连



b) CPU 断开与地址和数据总线的连接。DMA 控制器提供地址，外围设备驱动数据总线

图 4-30 普通周期和 DMA 周期中的数据流

DMA 控制的 I/O 与多处理器（multiprocessor）系统之间没有本质区别。DMA 控制器与

处理器相比还具有在外围设备和存储器之间拷贝数据块的指令集。可以认为DMA控制器就是具有两种指令的字符串处理器（string processor），这两条指令为MOVE string, Memory以及MOVE Memory, string。描述了数据传输策略后，下面将讨论影响点对点数据传输的一些因素（即术语和协议）。

4.4 I/O系统的性能

下面介绍I/O对计算机性能的影响。由于计算机要么执行计算要么访问外围设备，执行一个任务所需的总时间可表示为：

总时间 =CPU 时间 +I/O 时间

这个表达式过于简单，因为I/O操作通常可以与计算操作重叠。例如，多媒体系统可以从视频录像机上读取数据、处理以前读出的数据，以及将前面处理的数据存储至磁盘。

有些计算机应用（例如天气模拟）被称为CPU密集型（CPU intensive）的，因为它们需要更多的CPU时间和相对少的I/O时间。与之相反的应用称为I/O密集型（例如，从Internet上下载电影）。

即使是最快的磁盘驱动器相比CPU来说都要慢若干数量级。此外，处理器的性能比I/O子系统的性能增加得要快。例如，硬盘的访问时间多年来减小得很少，而处理器的速度一年内可能增加10%～40%。

计算机设计师对提高I/O的性能感兴趣出于以下几个原因。由于多媒体应用的发展，I/O瓶颈相对来说越来越重要。具有讽刺意味的是，人们可以容忍加载数据库、表格或文档的延迟，但是对在播放音乐或视频时即使是1/50s的短暂停顿也十分在意。

考虑以下这个虚构的例子。假定当前某多媒体应用程序是70%计算密集型的，使用计算机60%的I/O处理能力。假定改进后新版本的应用程序要求每年增加20%的资源（CPU和I/O）。计算机的处理性能每年增加40%，I/O的性能每年增加4%。在未来5年这种局势将如何发展？表4-3给出了这种局势随时间的变化。

表4-3 计算机与I/O能力和需求的相对增长

年份	CPU被使用的计算能力	I/O被使用的处理能力	CPU计算能力	I/O处理能力	CPU富裕计算能力	I/O富裕处理能力
0	70	30	100	50	30	20
1	84	36	140	52	56	16
2	100.8	43.2	196	54.08	95.2	10.88
3	120.96	51.84	274.4	56.24	153.5	4.4
4	145.15	62.21	384.16	58.49	239.01	无
5	174.18	74.65	537.82	60.83	363.64	无

在第二年，性能需求为84个单位的CPU处理能力和36个单位的I/O处理能力。在这段时间里，计算机发展了，CPU具有140个单位的计算能力，而I/O具有52个单位的处理能力。如读者所见，计算机的性能提升超过对计算性能的需求，所以没有问题。然而，对I/O系统性能需求的增加导致计算机能力逐渐落后于需求，直到再也不能运行该应用程序。

绝对、持续的数据传输速度并不是I/O系统性能中唯一的问题。相比于计算机系统中的其他地方，延时（latency）成为I/O系统性能的主要因素。以中断为例，延迟包括建立时间（中断调用和将返回地址保持在栈中）、中断处理时间以及中断返回（将返回地址弹栈并恢复

系统状态）时间。

假设计算机的中断建立和返回时间为 L（即中断开销），执行指令的平均速率为 p 条指令/s。如果中断处理程序具有 n 条指令，则中断系统的效率为：

$$E=\frac{\frac{100n}{p}}{L+\frac{n}{p}}=\frac{100n}{pL+n}$$

如果 L 的值大或者 n 的值小，中断机制的效率将很低。在几乎没有中断的系统中，这不是一个大问题。然而，在中断非常频繁的系统中，例如嵌入式实时控制器中，中断效率可能在系统设计中是一个主导因素。

假设计算机正在处理来自磁盘驱动器的块大小为 4096 字节的音频数据。从磁盘中读取数据的平均速率为 20Mb/s，每个数据块都需要的中断处理开销为 100μs。数据传输的效率如何？读取一块的时间为 4096×8×0.05μs=1638.4μs。每块都需要 100μs 的中断开销，对应的效率为 100×1638.4/(100+1638.4)=94.2%。

至此已经介绍了输入/输出的基础，下面将介绍数据是如何在内部子系统（存储器和 CPU）之间以及 CPU 与外围设备之间移动的。

4.5 总线

下面开始讨论总线，或者说总线系列（family of buses），它们是所有现代计算机的操作和性能的关键。总线（bus）这个术语是拉丁词“omnibus”的缩写，其含义为“为所有人（for all）”。总线就像高速公路一样可以由多个设备共享使用。在计算机中，所有希望相互通信的设备都使用总线。在介绍连接计算机和打印机等外部设备的串行总线系列之前，本节首先讨论计算机内部的高速内部总线。

图 4-31 显示了具有 3 条总线的计算机的组成。系统总线（system bus）由来自 CPU 的地址、数据和控制路径组成。存储器和存储器映射的 I/O 设备与该总线连接。这样的总线必须以其上最快的设备（通常是存储器）的速度进行操作。系统总线证明一刀切（one size fits all）的方法并不适用于计算机的设计，因为将低成本、低速外围设备与高速总线相连的成本效益比非常低。稍后将看到因为需要连接低成本外围设备而出现的低成本总线。

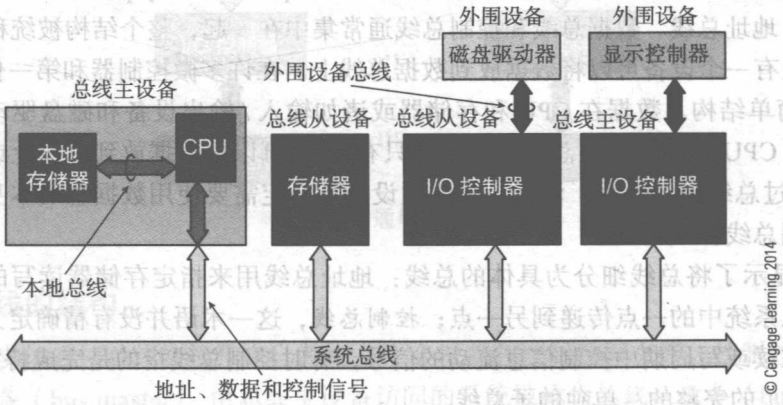


图 4-31 总线

本节将介绍总线的组织或拓扑结构（topology）；也就是，将描述它们如何将功能单元连接在一起，不同的总线之间是如何互连的。还将讨论使用总线在不同系统之间实现有意义的对话所需信号的作用，以及这些信号的协议和时序。

在具有多个 CPU 的系统中（或至少一个以上的设备可以像 CPU 一样启动数据传输动作（initiate data transfer action）），总线必须决定在需要访问总线设备中，哪个设备应该获得访问保障。这种机制被称为仲裁（arbitration），它是现代系统总线的关键特性。本节将介绍总线仲裁的基本原理和实现总线仲裁的方法。

可以控制系统总线的设备被称为总线主设备（bus master），只能对远程总线主设备发起的事务进行响应的设备称为总线从设备（bus slave）。在图 4-31 中，CPU 为总线主设备，存储器为总线从设备。图 4-31 中某个 I/O 端口被标识为总线主设备，这是因为它可以控制总线（如，DMA 数据传输），而其他的外围设备被标识为总线从设备，这是因为它只能对读或写访问进行响应。磁盘驱动器和其控制器之间的连接也被标识为总线（bus），这是因为它是一个代表特殊和高度专用总线的例子。在图中的一个方框中显示了带有本地存储器和本地总线（local bus）的 CPU。某些计算机系统是这样组织的，CPU 通过本地总线与位于相同卡（card）上的存储器系统直接连接。这种组织方式意味着 CPU 访问数据时并不是全部都需要使用系统总线。

前面已经指出个人计算机崛起的原因之一是其可以很方便地与各种不同的系统相连，例如高速硬盘、视频摄像机、键盘或者鼠标。因此，本节将介绍一些外围设备总线，它们为处理器与外部子系统相连进行了专门的设计。这些总线包括 SCSI 总线、火线（Firewire）和以太网（Ethernet）。

本节还将简要提到计算机总线系列中的一位无名英雄：即插即用（plug-and-play）。20 世纪 70 ～ 80 年代，将新的外围设备与计算机相连是名副其实的噩梦。用户不得不关心物理连接的详细信息、计算机的硬件配置以及外围设备的硬件配置。用户必须指定外围设备的地址空间、中断请求号、DMA 通道、握手机制等等。今天，人们认为类似即插即用等技术是理所当然的，这些技术可以使计算机和外围设备自动协议，将资源分配给外围设备并使资源不与任何其他外围设备的资源冲突。

4.5.1 总线结构和拓扑

图 4-32 说明了由 CPU+ 存储器 + 本地总线（CPU plus memory plus local bus）组成的简单总线结构。地址总线、数据总线和控制总线通常集中在一起，整个结构被统称为总线（the bus）。同时只有一个设备可以将数据放到数据总线上。在许多微控制器和第一代微机中可以发现这样的简单结构。数据在 CPU 和存储器或诸如输入 / 输出设备和磁盘驱动器的外围设备之间传输。CPU 可能永远是总线主设备，只有 CPU 可以把数据放到总线上或邀请存储器 / 外围设备通过总线提供数据。注意，总线主设备不一定需要使用数据总线本身；它可以像代理那样控制总线。

图 4-32 显示了将总线细分为具体的总线：地址总线用来指定存储器读写的位置；数据总线将数据从系统中的一点传递到另一点；控制总线，这一术语并没有精确定义，有时控制总线指的是在读或写周期中控制信息流动的信号，有时控制总线指的是完成特定功能（如中断控制或仲裁）的完整的、单独的子总线。

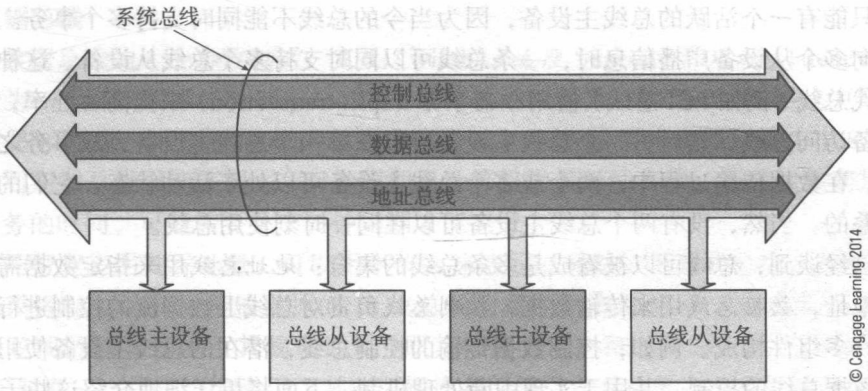


图 4-32 通用总线的结构

高性能计算机系统，如 PC，具有比图 4-31 更复杂的总线结构。图 4-33 展示了将两条总线通过扩展接口（expansion interface）连接起来的总线结构。这两条独立的总线系统可能具有完全不同的功能：一条可以优化高速处理器/存储器事务，另一条可以支持大量可接插的外围设备。PC 的扩展总线本来是想允许低成本的外围设备插入 PC 来创建开放的系统，并提供灵活性。

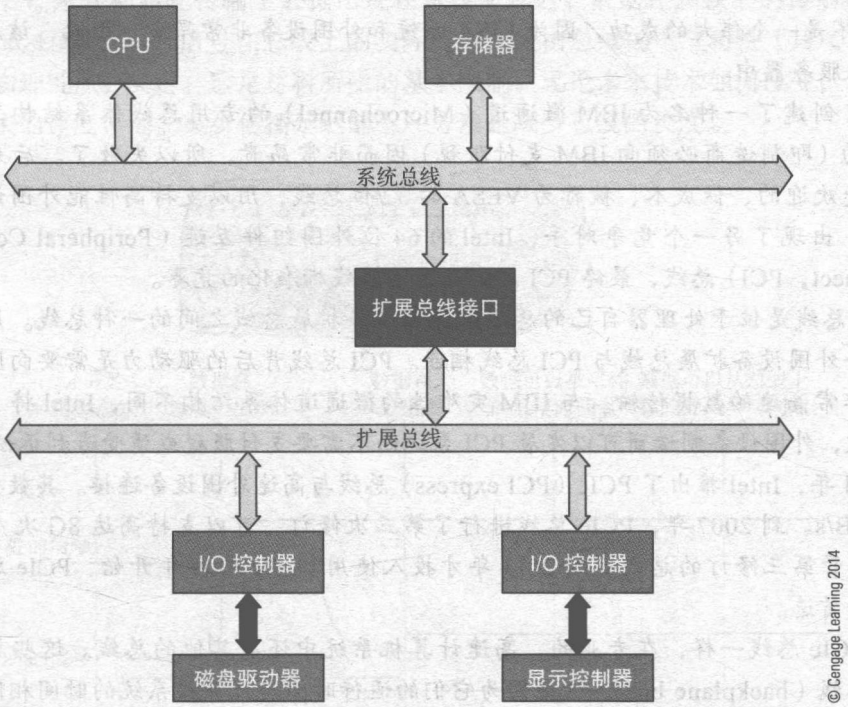


图 4-33 通用总线的结构

4.5.2 总线的结构

当两个实体通过总线交换信息，其中一个会发起和控制数据交换。控制总线的系统被称为总线主设备（bus master），由总线主设备访问的系统被称为总线从设备（bus slave）。在任

何时刻，只能有一个活跃的总线主设备，因为当今的总线不能同时支持多个事务。当总线主设备需要向多个从设备广播信息时，一条总线可以同时支持多个总线从设备。这种描述过于简单。现代总线（例如PCI总线）使用分离事务（split transaction）来提高吞吐率，其中一个总线主设备访问总线，然后另一个总线主设备可以在第一个总线主设备完成事务之前使用总线。因而，在数据传输过程中，两个或多个总线主设备可以处于活动状态，它们的数据传输过程是重叠的。当然，没有两个总线主设备可以在同一时刻使用总线。

前面已经谈到，总线可以被看成是多条总线的集合：地址总线用来指定数据需要传输的源或目的地址，数据总线用来传输数据，控制总线负责对总线上数据流的控制进行响应。控制总线由许多组件构成。例如，控制数据传输的控制总线，潜在的总线主设备使用它以获得对地址和数据总线的控制，并用于实现中断处理机制。下面将更详细地介绍这些子总线。

PC总线的历史

最早的PC扩展总线提供了8位数据通路，其工作在现在被认为少得可怜的4.77MHz的时钟速度下。后来在PC的AT版本引入了16位被称为ISA总线的扩展总线，其使用8.33MHz的时钟。对于专业的应用程序来说，这还不够快，所以提出了另一种扩展总线——扩展工业标准体系结构（extended industry standard architecture，EISA）总线。EISA支持32位数据通路，具有33MB/s的带宽，其时钟频率与ISA总线的相同。EISA并不是一个伟大的成功，因为EISA系统和外围设备非常昂贵。因此，该总线主要用于文件服务器中。

IBM创建了一种名为IBM微通道（Microchannel）的专用总线体系结构，由于它是专有的（即制造商必须向IBM支付版税）因而非常昂贵，所以失败了。后来出现了一种更受欢迎的、低成本、被称为VESA的32位总线，用以支持高性能外围设备。此后不久，出现了另一个竞争对手，Intel的64位外围组件互连（Peripheral Component Interconnect，PCI）总线，最终PCI总线赢得了总线标准化的竞赛。

PCI总线是位于处理器自己的总线和外围设备扩展总线之间的一种总线。用户可以将第二个外围设备扩展总线与PCI总线相连。PCI总线背后的驱动力是需要向图形控制器进行非常高速的数据传输。与IBM灾难性的微通道体系结构不同，Intel将PCI总线标准开放，外围设备制造商可以发展PCI系统而不需要支付版税或遭受专利诉讼。

2004年，Intel推出了PCIe（PCI express）总线与高速外围设备连接。其数据传输率为250MB/s。到2007年，PCIe总线进行了第三次修订，可以支持高达8G次/s的传输速率（尽管第三修订的总线直到2011年才投入使用）。从2009年开始，PCIe总线成为图形卡的首选。

与PCIe总线一样，在专业的、高速计算机系统中还有其他的总线。这些总线被称为背板总线（backplane bus），这是因为它们的运行时间与计算机系统的时间相同且所有模块（电路卡）都插在它上面。与PC总线已经成为连接CPU的主板的一部分不同，这些其他类型的总线很少或没有有源电路，CPU就像其他存储器或外围设备那样插入总线。典型的背板总线包括VMEbus、Multibus、NuBus以及Futurebus+。

多个处理器可以通过背板总线连接在一起，甚至可以把它理解为一种CPU的局域网。连接至这种总线的模块或卡常常包含其自己的本地总线以连接本地存储器和外围设备。

1. 数据总线

定义数据总线的3个参数包括：宽度（width）、速度（speed）和延迟（latency）。数据总线的宽度为一个总线周期（bus cycle）内可以传输的比特数。20世纪70年代中期，典型的数据总线是8位宽，而今天64位的总线非常普遍。一些芯片的内部总线为128位或更宽。注意时钟周期和总线周期不同——时钟周期是计算机中最短的事件，而总线周期是总线上一次完整事务的时间，可能需要几个时钟周期才能完成。

总线的带宽表明其吞吐量，可以用字节/s来表示。显然，更宽的总线，吞吐量就越大。例如，如果将以100MB/s传输数据的16位总线的宽度加倍为32位，它将形成200MB/s的吞吐量。总线的固有速度由其物理性质（即其构建方法）和连接到总线上的设备（见后文“在总线上跳跃——信号的传输”）。某些逻辑设备能够以比其他设备更高的速度操作。

总线的延迟为设置数据传输所花费的时间。在某设备永远为总线主设备时总线的延迟可能会很低，但是在需要传输数据的设备必须等待仲裁机制保障其总线使用权这样的系统中，就会具有长得多的延迟。

2. 总线的速度

假设设备A在总线的一端传输数据至总线的另一端。下面来看看设备A在t=0时刻初始化数据传输后发生的一系列事件（如图4-34所示）。最初，A在t_d时刻将数据放到数据总线上，其中t_d为A初始化传输至数据出现在总线上延迟。数据在总线上的传播速度约为光速的70%或1英尺/ns。信号在总线上的实际传输速度由总线的电气特性（其尺寸和导体周围物质的物理性质）决定。这是材料所受的基本限制，无论未来技术如何改变，信号都不能以比光更快的速度传播。减少传播延迟的唯一方法是减少总线的长度。

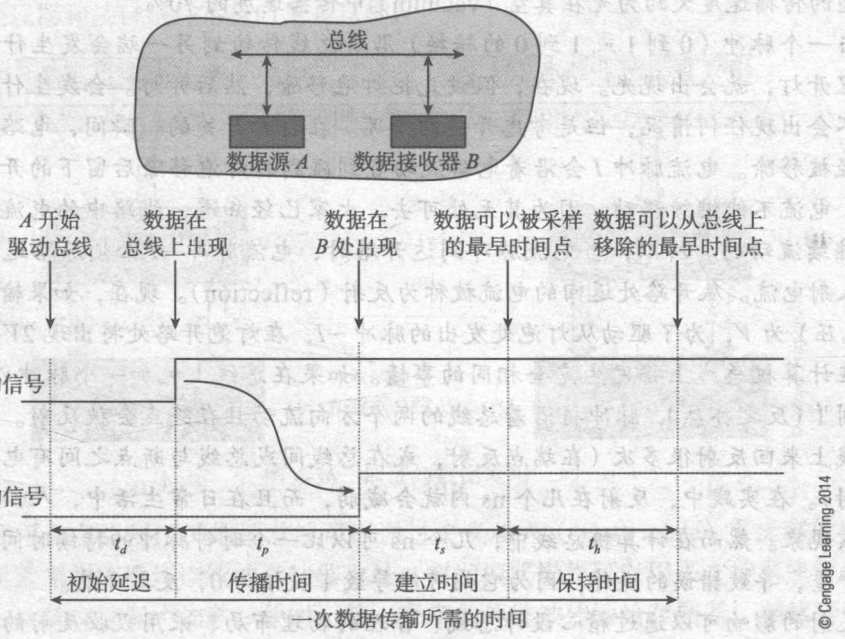


图4-34 传输时序

当数据达到B时，它必须被捕获（captured）。锁存和存储器部分由其建立（setup）时间和保持（hold）时间指定。数据建立时间t_s是数据在系统B的输入端出现至其被识别所花费

的时间。数据保持时间 t_h 指的是数据被捕获后在系统 B 的输入端保持稳定所需的时间。

因此数据传输所花费的时间 $t_T=t_d+t_p+t_s+t_h$。代入典型的参数值可得 $t_T=4+1.5+2+0=7.5$ns，对应的数据传送频率为 $1/7.5\text{ns}=10^9/7.5=133.3$MHz。32 位宽的总线的最大数据传输率为 533.2MB/s。在实践中，数据传输需要时间来启动，该时间称为延迟 t_L。将延迟考虑在内，最大数据传输率为 $1/(t_T+t_L)$。

通过流水线（pipelining）技术能够达到更高的数据传输率，该技术在系统 B 完成读取前一个数据之前开始发送下一个数据元素。图 4-35 显示了将流水线应用于上一个例子的情况。数据必须在系统 B 的输入端稳定至少 t_s+t_h 秒；然后新的元素才可以取代以前的元素。流水线技术允许的终极数据传输率为 $1/(t_s+t_h)$。

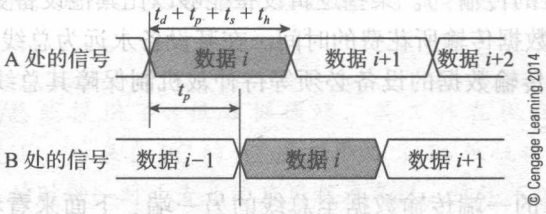


图 4-35 数据传输流水

在总线上跳跃——信号的传输

常见的误解是，总线上信号是以光速进行传播的。在所有的物理介质中，信号传播的速度是由导体的几何形状和将其隔离的材料（所谓的绝缘体）来决定的。信号在总线上典型的传播速度大约为光在真空（vacuum）中传播速度的 70%。

当一个脉冲（0 到 1 或 1 到 0 的转换）沿着总线传输到另一端会发生什么？如果在房间里开灯，就会出现光。现在，假设先把灯泡移除，然后开灯。会发生什么情况呢？似乎不会出现任何情况，但是考虑开关的情况。在打开开关的一瞬间，电路并不知道灯泡已经被移除。电流脉冲 I 会沿着电路流动直到碰到将灯泡移除后留下的开路。在此处以后，电流不能继续移动，因为其无处可去。大家已经知道，线路中的电流在稳态下是不能继续流动的。所以，当电流脉冲到达开路时，电流脉冲 $-I$ 必须原路返回以取消输入或入射电流。从开路处返回的电流被称为反射（reflection）。现在，如果输入脉冲的幅度（电压）为 V，为了驱动从灯泡处发出的脉冲 $-I$，在灯泡开路处将出现 $2V$ 的电压。

在计算机总线上将发生完全相同的事情。如果在总线上施加一个脉冲以驱动总线，从 0 到 1（反之亦然），脉冲将沿着总线的两个方向流动且在终点会被反射。该脉冲可以在总线上来回反射很多次（在端点反射，或在总线间或总线与断点之间有电阻改变的地方反射）。在实践中，反射在几个 ns 内就会减弱，而且在日常生活中，人们通常不会意识到该现象。然而在计算机总线中，几个 ns 可以比一个时钟脉冲的持续时间要长，反射会很严重，导致错误的操作，因为它们可能导致 1 被读为 0，反之亦然。

反射的影响可以通过精心设计总线、合理的物理布局、采用吸收反射的终端设备来减小。如果负载（电阻）连接在总线的终点、负载的阻抗与总线的特性负载相同，就不会发生反射。总线的特性阻抗通常在 50 ～ 200Ω 之间。反射系数 Γ（脉冲中被反射部分所占比例）可由以下方程给出，其中 Z_L 为终端电阻而 Z_S 为特性阻抗。

$$\Gamma = \frac{Z_L \quad Z_S}{Z_L \quad Z_S}$$

1858 年，铺设了第一条横跨大西洋的电缆，长度超过 3000 英里，信号传输速率远低于期望的结果（第一根电缆在发出 400 条消息后就坏了，直到 1886 年，使用了一条新电缆）。为什么单根电缆传输速率出人意外地低呢？这个问题被交给 William Thomson 教授来调查。他建立了传输线理论和总线脉冲行为的基础。读者可以认为这就是电子技术的开始。一开始由 Thomson 教授领导该工作，后来变成由 Lord Kelvin 领导。

3. 地址总线

当 CPU 访问存储器时，控制数据传输的总线主设备必须提供数据的源或目标地址（address）。某些计算机系统有明确的、与数据总线并行工作的地址总线。例如，当处理器向存储器写数据，32 位的地址总线在将数据传递给数据总线的同时将地址传递给地址总线。某些系统将地址和数据总线组合在一起形成多路复用的地址/数据总线来传输地址和数据（其中数据和地址交替使用总线）。由于时间被划分为地址槽和数据槽，因此这样的总线被称为时分（time-division）多路复用。

图 4-36 描述了更便宜的多路复用地址/数据总线，由于它需要更少的信号通路和连接器，而且插座需要更少的引脚，其实现比传统的非多路复用总线要便宜。将相同的信号线多路复用为地址和数据需要在线路的一端使用一个多路复用器（multiplexer，一种高速电子开关），且在另一端需要使用多路选择器（demultiplexer）。多路复用总线的速度要比非多路复用总线的速度慢，通常在更看重成本而不太看重速度时使用。尤其是当多路复用器和多路选择器内置于处理器和接口组件本身时更是这样。

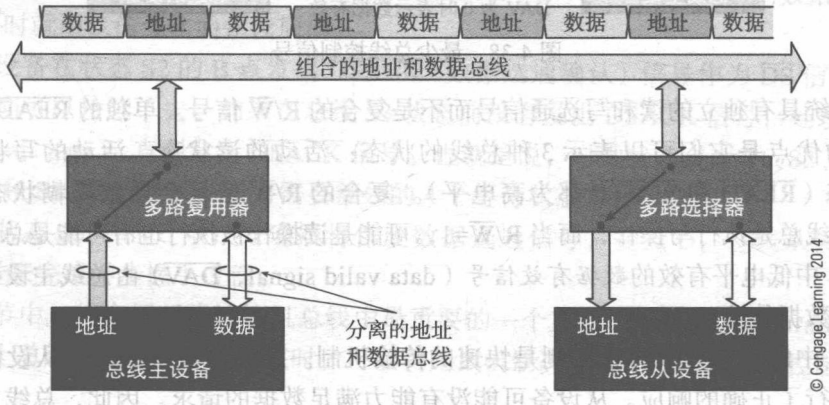


图 4-36 地址与数据复用

非多路复用和多路复用的地址总线可以通过突发模式（burst mode）来提高效率，此时可以把数据元素序列传输到连续的存储器地址。突发模式操作可以用来支持高速缓冲存储器系统。当 Cache 从存储器中加载一块时，第一个字的地址被传输给存储器。存储器根据指定地址提供数据，紧随其后的就是下一个地址的下一个字，等等。这些连续的地址可以由存储器生成。

图 4-37 展示了突发模式寻址的概念，访问地址 i 并传输数据 i；而数据 i+1、i+2 和 i+3 并不需要进一步提供地址就可以传输。

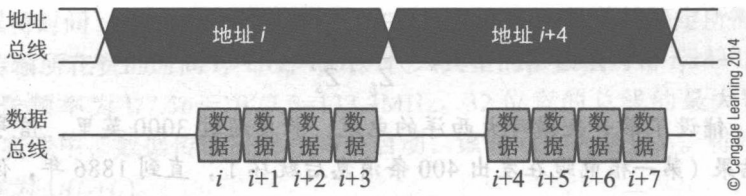


图 4-37 突发模式和数据

4. 控制总线

控制总线控制数据总线上的信息流。图 4-38 描述了最简单的由两条线构成的同步[⊖]（synchronous）控制总线，这两条线分别为*数据方向*（data-directional）信号和*数据有效*（data validation）信号。数据方向信号通常被称为 R/$\overline{W}$，高电平表示读操作，低电平表示写操作（数据传输的方向由发起数据传输的总线主设备指定）。在读周期中，数据流从总线从设备流向主设备；在写周期中，数据从总线主设备流向从设备。

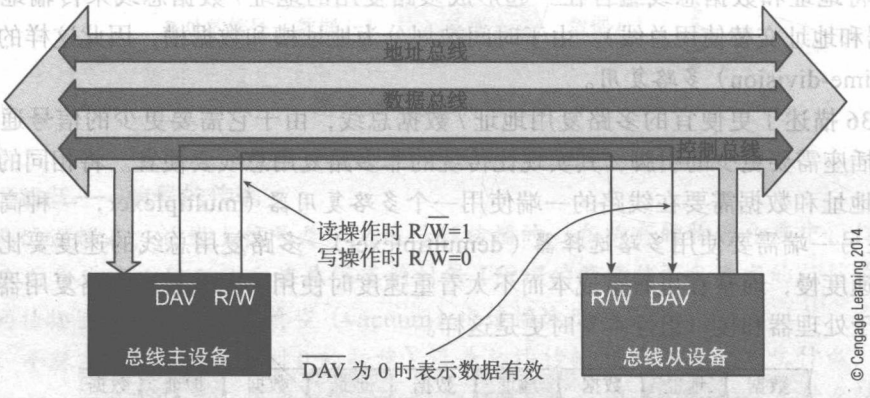


图 4-38 最少总线控制信号

有些系统具有独立的读和写选通信号而不是复合的 R/$\overline{W}$ 信号。单独的 $\overline{READ}$ 和 $\overline{WRITE}$ 信号具有的优点是它们可以表示 3 种总线的状态：活动的读状态、活动的写状态以及总线空闲状态（$\overline{READ}$ 和 $\overline{WRITE}$ 都为高电平）。复合的 R/$\overline{W}$ 信号会导致模糊状态，因为当 R/$\overline{W}$=0，总线总是执行写操作，而当 R/$\overline{W}$=1，可能是读操作被执行也有可能是总线空闲。

图 4-38 中低电平有效的*数据有效信号*（data valid signal，$\overline{DAV}$）由总线主设备发出，指明开始进行数据传输。

图 4-38 中的同步开放传输机制是快速的传输机制。总线主设备不知道从设备是否对其访问请求进行了正确的响应。从设备可能没有能力满足数据的请求。因此，总线主设备必须以参与数据传输从设备的最慢速度操作。本书讨论的异步总线将通过*握手*（handshaking）机制来解决上述两个问题。

异步传输数据需要两个数据流控制信号：数据选通信号表明数据已经准备就绪，数据确认选通信号表明数据已被读取。前文曾谈到，总线事务有时不能完成（例如，主设备可能访问一个不存在的存储器位置），数据有效选通信号永远不会得到确认。当这种情况发生时，主设备必须在一个合适的超时时间到达后终止事务本身。通常，当主设备发出数据有效信号

⊖ 在需要主时钟来同步信号时，总线就是同步的。例如，总线主设备假定从设备在数据有效信号发出后已经响应了 *n* 个时钟周期。

时，启动定时器。如果定时器达到预定值，主设备将被迫放弃访问。处理器执行一个称为总线错误（bus error）的特殊例外来调用操作系统。

下面看一个异步数据传输的例子——处理器存储器读周期。图 4-39 给出了 68020 处理器的简化读周期时序图。处理器由时钟（CLK）控制，最小的总线周期为有 6 个时钟状态，标记为 S0 ～ S5。

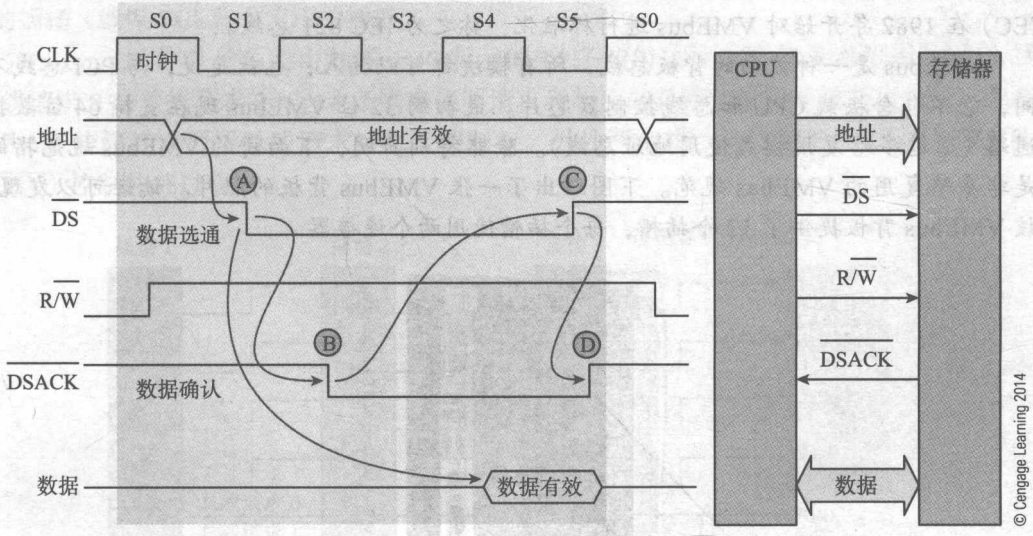


图 4-39　异步数据传输的例子

一旦处理器在 S0 状态将地址放到地址总线上，CPU 在 S1 状态的 A 点发出数据选通（data strobe，$\overline{DS}$）信号触发数据传输。如果总线从设备（如，存储器）检测到数据选通信号，当 R/$\overline{W}$ 为高时就用该信号初始化读周期。

总线从设备在状态 S2 的 B 点发出 $\overline{DSACK}$（数据选通确认）信号作为 $\overline{DS}$ 信号的响应，表明总线周期可以持续下去。如果在总线 S3 状态前没有发出 $\overline{DSACK}$ 信号，处理器将插入等待状态（wait state），直到发出 $\overline{DSACK}$ 信号。也就是说，时钟状态会为 S1、S2、W、W、W、W、S3、S4 和 S5（假设总线从设备需要额外的 4 个等待状态来完成访问）。

在总线状态 S5 中的 C 点，处理器将作废数据选通信号，在点 D，总线从设备将作废 $\overline{DSACK}$ 信号来完成异步握手过程。

在下一节中，将介绍现代计算机总线中最重要的一个方面——仲裁（arbitration），当两个或两个以上总线主设备争夺控制权时，可以利用该机制决定哪个主设备可以控制总线。

4.6　总线仲裁

若某系统中有多个潜在的总线主设备通过公共总线连接，则需要一种机制来处理同时发出的总线请求。多个请求被识别且赋予其中一个优先使用权的过程叫作仲裁（arbitration）。

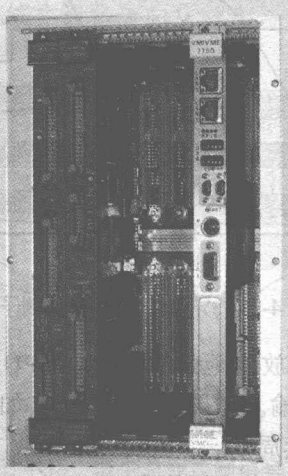
有两种方法来处理多个总线请求：本地化（localized）仲裁和分布式（distributed）仲裁。在本地化仲裁中，仲裁电路接收来自竞争的总线主设备的请求，然后决定谁来控制总线。在分布式仲裁的系统中，每个主设备将参加仲裁过程，且系统缺乏特定的仲裁者——每个主设备监控其他的主设备，然后决定是继续竞争使用总线还是放弃并等待一段时间。本节将描述 3 种实现这些技术的总线——用于高性能专业（如，工业控制）系统中的 VMEbus、用于

Apple Macintosh 和几个专业系统中的 NuBus，以及 PCI 总线。

背板总线

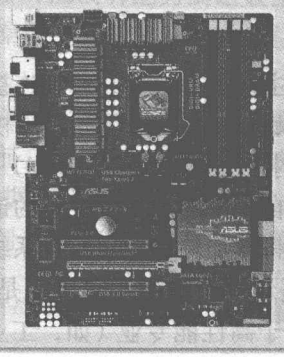
VMEbus 是一种异步的、非复用的背板总线，最初由 Motorola、Signetics、Mostek、Philips 和 Thomson-EFCIS 公司支持，并迅速被接纳为工业标准。1984 年，IEEE 批准 VMEbus 为 IEEE 1014 总线。国际电工委员会（International Electrontechnical Commission，IEC）在 1982 年开始对 VMEbus 进行标准化，称之为 IEC 821 总线。

VMEbus 是一种被动的背板总线，所有模块都可以插入；也就是说，与 PCI 总线不同，它不包含板载 CPU 和总线控制器芯片。最初的 32 位 VMEbus 现在支持 64 位数据通路（通过多路复用模式使用地址总线）。除非特别声明，下面讲的 VMEbus 规范指的是非多路复用的 VMEbus 规范。下图给出了一张 VMEbus 背板的照片。读者可以发现，该 VMEbus 背板提供了 12 个插槽，每个插槽使用两个连接器。



http://en.wikipedia.org/wiki/File:VMEbus.jpg. Courtesy of Sergio Ballestrero.

下图显示的是某些 PC 中的现代高性能主板。读者可以发现，大部分的计算机硬件位于主板上。当 PC 首次出现时，主板只是简单地将功能模块相连；如果需要音频输入/输出，用户需要购买声卡，然后插进主板就可以了。甚至有必要为简单的串行数据链接购买接口。图中的这块主板，使用 Intel 的 Z77 芯片组，适用于高性能系统。它包含完整的音频接口、高速以太网链接、甚至提供支持 HDMI 所必需的逻辑电路。板上具有 PCI、PCIe 扩展槽、USB 3.0 接口以及能够支持 32GB 的 DDR3 DRAM 的插槽。主板还提供了 8 个支持外部存储器（包括 SSD、磁盘以及光盘）的串行 SATA 接口，其中 4 个具有 6GB/s 的通信能力。



Alan Clements

4.6.1 本地化仲裁和 VMEbus

本节将介绍设备如何请求总线和总线如何保障这个请求。这里使用支持几种功能模块类型的 VMEbus 作为例子。本节关注控制总线的总线主设备（bus master）、请求总线的总线请求者（bus requester）和将总线使用权授予总线主设备的仲裁者（arbiter）。总线请求者受雇于希望访问 VMEbus 的总线主设备。VMEbus 通常置于一个盒中，该盒具有一些模块可以插入的插槽（就像 PCI 总线使用的插槽）。

VMEbus 的仲裁子总线由如图 4-40 所示的 14 条线组成。总线请求者使用 $\overline{BR0}$ ～ $\overline{BR3}$（总线请求 0 ～总线请求 3）这 4 条线来表明请求者相应的总线主设备希望使用总线。仲裁者使用 4 条应答线来授予总线请求者总线控制权。另外两个线，总线清除（$\overline{BCLR}$）和总线忙（$\overline{BBSY}$）控制仲裁过程。

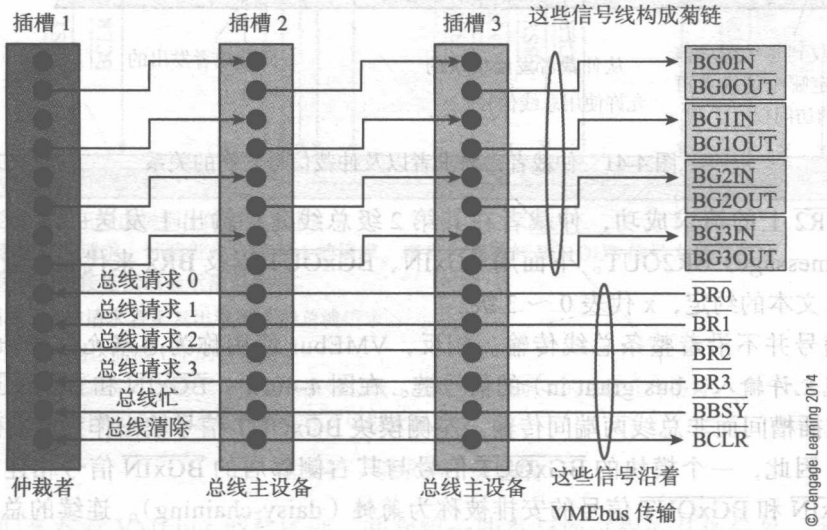


图 4-40 VMEbus 仲裁信号

VMEbus 仲裁者位于 VMEbus 称为插槽 1 的特殊（special）位置上。VMEbus 的第一个位置必须由仲裁者占用。所有的总线请求线沿着 VMEbus 传输，任何的主设备可以将请求放在这些线之一上。总线请求者发出请求的特定级别由用户定义；也就是说，用户决定 4 个总线请求线与模块的请求输出相连。

仲裁者沿着总线插槽读取所有的总线请求输入，决定服务哪个请求，然后通过总线应答输出通知其他模块其决定。在介绍完仲裁总线如何操作后，再介绍仲裁者决定哪个竞争使用总线的主设备获得总线的方式。

VMEbus 支持 4 个级别的仲裁。下面很快就会看到，这四级中的每一级都可以进一步细分。总线请求信号沿着 VMEbus 传输，在插槽 1 的仲裁者处终止。图 4-41 展示了这些线、仲裁者、一级总线请求者模块之间的关系。

当一个或多个总线请求者代表其相应的总线主设备希望访问 VMEbus 时，它们向其分配的请求线发出请求。例如，插槽 3 中的卡可能向总线请求线 $\overline{BR1}$ 发出信号，插槽 5 中的卡可能向总线请求线 $\overline{BR3}$ 发出信号。插槽 1 中的仲裁者检查所有的总线请求信号输入，然后决定是否其中之一请求成功。

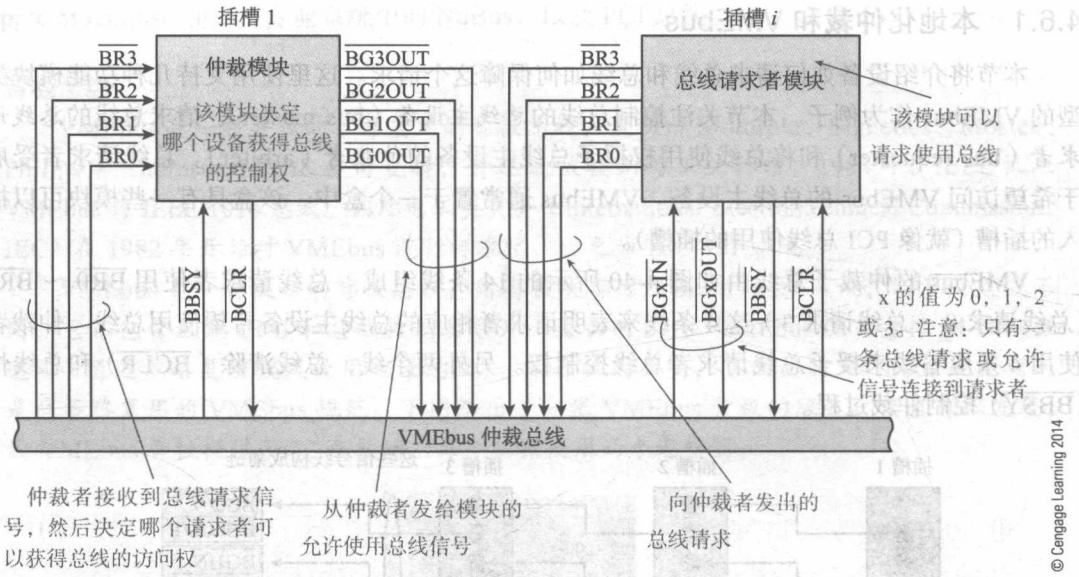


图 4-41 仲裁者、请求者以及仲裁信号之间的关系

假如 $\overline{BR2}$ 上的请求成功，仲裁者在其第 2 级总线允许输出上发送一个总线允许消息（bus grant message）$\overline{BR2OUT}$。下面用 $\overline{BGxIN}$、$\overline{BGxOUT}$ 以及 $\overline{BRx}$ 来代表 x 级的信号。按照 VMEbus 文本的约定，x 代表 0 ～ 3 级。

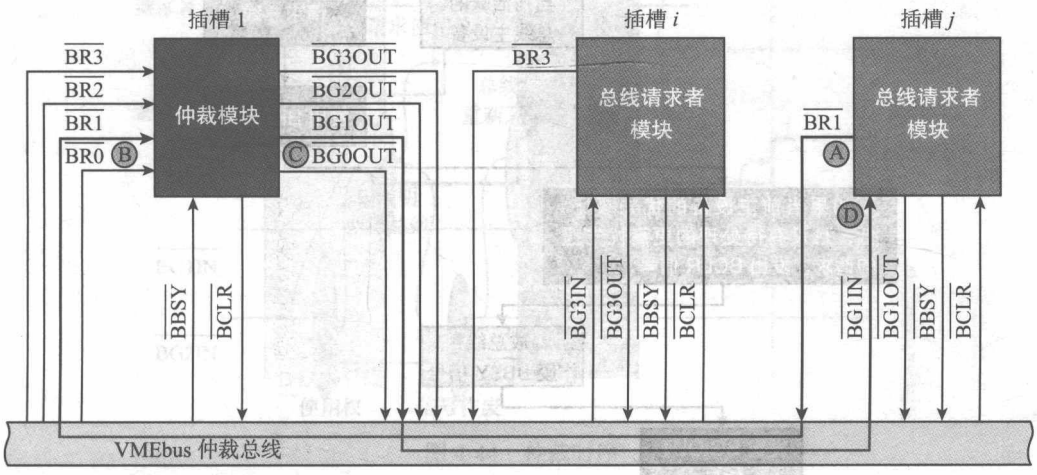
$\overline{BGx}$ 信号并不沿着整条总线传输。相反，VMEbus 使用称为总线允许输出（bus grant out）和总线允许输入（bus grant in）的信号链。在图 4-40 中，$\overline{BGxIN}$ 和 $\overline{BGxOUT}$ 信号线是分开的，在插槽间而非总线两端间传输。左侧模块 $\overline{BGxOUT}$ 信号输出作为其右侧的 $\overline{BGxIN}$ 信号输入。因此，一个模块的 $\overline{BGxOUT}$ 信号与其右侧邻居的 $\overline{BGxIN}$ 信号相连接。图 4-40 中这种 $\overline{BGxIN}$ 和 $\overline{BGxOUT}$ 信号的安排被称为菊链（daisy-chaining）。连续的总线信号线在与其连接的所有设备间按照两个方向传输。菊链信号线为单向的（unidirectional），沿着一个特定方向从一端传输到另一端。接入菊链的每个模块（即从其中接收和向其发出信号的模块）要么向下传递一个信号，要么向其发出一个信号。

图 4-42 显示了这样一个系统，其插槽 j 的请求者在没有其他设备请求总线时发出 1 级总线请求。当发出 $\overline{BR1}$ 时，插槽 1 中的仲裁者检测到该信号并发出 $\overline{BG1OUT}$ 信号，该信号一直传递下来，直到到达插槽 j。插槽 1 中的仲裁者发出总线允许输入信号至插槽 2 中的卡。插槽 2 中的卡收到总线允许输入然后将其通过总线允许输出传递给插槽 3 中的卡，以此类推。这样，一个卡从其左邻居收到总线允许输入，然后将其以总线允许输出信号传递给其右邻居。然而，某卡可以选择终止菊链信号传递序列，不将总线允许信号传递给其右邻居，后面马上会看到。如果某插槽因为没有卡插入是空的，需要提供总线跳线（即链路）使得合适的 $\overline{BGxIN}$ 传递给相应的 $\overline{BGxOUT}$ 端。

请求者模块通过在总线请求线 $\overline{BR0}$ ～ $\overline{BR3}$ 之一上发出信号来试图控制系统数据传输总线。只有一条请求线发出信号，通过赋予请求者一个给定的优先级来选择实际的信号线。该优先级可以通过板上用户可选的跳线赋值或者软件来动态选择。

在插槽 1 中仲裁者，在接收到总线请求后，在某条 $\overline{BGxOUT}$ 线上发出信号，然后总线允许信号沿着菊链传播（允许信号的级别与赢得当前一轮仲裁的请求的级别相关）。每个

$\overline{BGxOUT}$ 信号将到达下一个模块的 $\overline{BGxIN}$。如果该模块不需要访问总线，它将该信号传递给其 $\overline{BGxOUT}$ 线。然而，如果该模块希望请求总线，它不发出其 $\overline{BGxOUT}$ 信号，自己接管总线的控制。菊链提供了自动优先级，因为后面的总线请求者接收不到总线允许信号——这被称为地理优先级（geographic priotitization）。



插槽 j 中的请求者正发出级别 1 的总线请求。
没有待处理的其他请求，仲裁者允许级别 1 的请求。来自仲裁者的 $\overline{BG1OUT}$ 信号为菊链传输。
图中用蓝色表示直接连接以显示消息通路。

Ⓐ 插槽 j 中的请求者正发出级别 1 的总线请求。
Ⓑ 总线请求到达插槽 1 中的仲裁者。
Ⓒ 仲裁者决定级别 1 的总线请求是否获得总线，然后发出级别 1 的总线允许信号。
Ⓓ 插槽 j 中的请求者收到总线允许信号，可以控制总线。

© Cengage Learning 2014

图 4-42　请求第 1 级总线

下面我们来看看 VMEbus 仲裁序列，并介绍一些参与该过程的其他信号。图 4-43 提供了 VMEbus 仲裁过程的协议流程图（protocol flowchart），其中描述了一次仲裁过程中的一系列事件。

图 4-43 中，一开始，总线插槽 M 中优先级小于 i 的总线主设备正在控制总线。当前总线主设备发出总线忙（$\overline{BBSY}$）信号，该信号沿着总线传递。只要任何一个主设备发出了 $\overline{BBSY}$ 信号，其他的主设备都不能试图获得 VMEbus 的控制权，即不能强迫 VMEbus 上处于活跃状态的总线主设备放弃总线控制权。

假设在插槽 N 中的总线请求者发出级别为 i 的总线请求。该请求级别比当前主设备的要高。仲裁者检测到新的更高级别的请求，则输出信号 $\overline{BCLR}$（总线清除）。来自仲裁者的总线清除信号通知当前在插槽 M 中的主设备，另一个具有更高优先级的主设备现在希望访问总线。当前主设备不必在规定时间限制内放弃总线。通常情况下，它将在方便的时候通过作废 $\overline{BBSY}$ 信号来释放总线。注意，VMEbus 提供了由菊链中的插槽位置决定的地理优先级和由总线请求级可选的优先级（马上将会看到）。

$\overline{BCLR}$ 由向总线请求信号线永久性地被赋予固定（fixed）优先级的仲裁者驱动。其他仲裁机制，如后面将介绍的轮转仲裁机制，没有固定的优先级，仲裁者不需要使用总线清除信号。

当仲裁者检测到当前主设备已经释放了总线，仲裁者发出 $\overline{BGiOUT}$ 信号表示级别为 i 的请求者已经获得总线控制权。仲裁者只知道请求的级别，而不知它来自哪个插槽。总线允许

信号沿着总线传输，通过 $\overline{BGiIN}$ 进入，通过 $\overline{BGiOUT}$ 输出。当该消息到达插槽 N 中发出级别 i 请求的请求者处，消息不再继续传递。

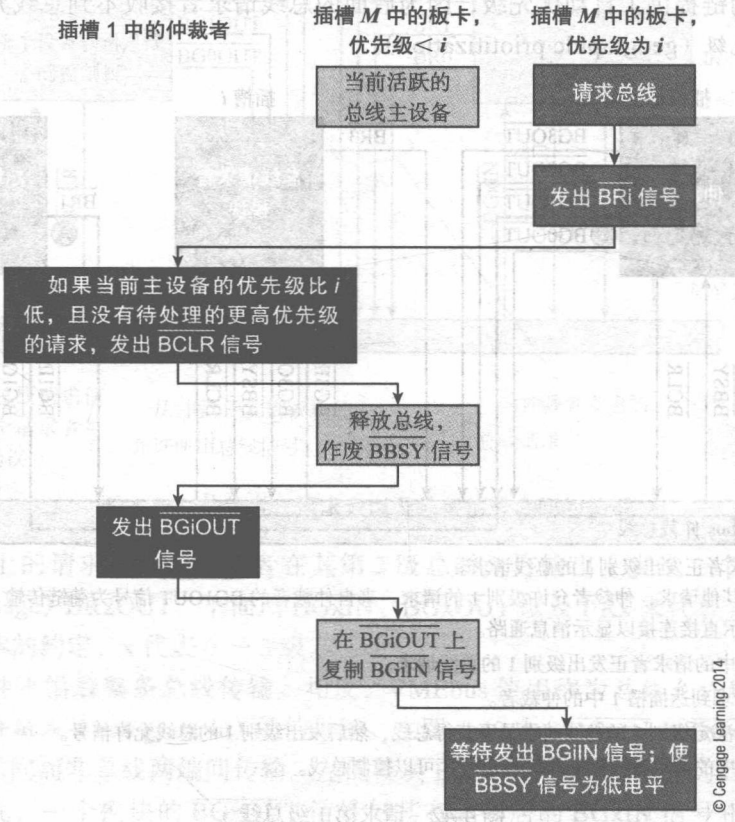


图 4-43 VMEbus 仲裁协议流程

相反，请求者发出 $\overline{BBSY}$ 信号表明其当前正在控制总线。如果有一个位于比插槽 N 离仲裁者更近的、级别为 i 的请求者也几乎在同一时间请求总线会出现什么情况？答案是，离仲裁者更近的请求者将首先收到总线允许信号并控制总线。

1. 释放总线

要释放总线，请求者可能有两个选项：用完释放（release when done，RWD）或者根据请求释放（release on request，ROR）。选择 RWD 要求请求者在主设备停止指示总线忙后尽快释放总线；也就是说，主设备在其任务已经完成后还可能控制总线，从而导致不适当的总线占用。ROR 选项更适合不会允许赋予主设备无限的总线访问权这样的系统。ROR 请求者检测 4 条总线请求线。如果它看到另一个请求者已经请求了服务，它就会发出 $\overline{BBSY}$ 信号并被其他请求延迟。由于总线经常被主动清除，ROR 选项也减少了主设备请求仲裁的次数。

2. 仲裁过程

下面看一个仲裁程序的例子。图 4-44 展示了当两个具有不同优先级的请求者请求总线时会发生什么事情。一开始，请求者 A 和 B 同时发出其总线请求。请求者 A 发出 $\overline{BR1}$ 信号，请求者 B 发出 $\overline{BR2}$ 信号。假设仲裁者检测到 $\overline{BR1}$ 和 $\overline{BR2}$ 都是低电平，由于 $\overline{BR2}$ 比 $\overline{BR1}$ 具有更高的优先级，仲裁者将在插槽 1 中发出 $\overline{BG2IN}$ 信号。当总线允许信号沿着菊链传递给请求者 B 时，请求者 B 将发出 $\overline{BBSY}$ 信号以响应 $\overline{BG2IN}$。然后，请求者 B 释放

$\overline{BR2}$，并通知自己的主设备现在可以使用 VMEbus。

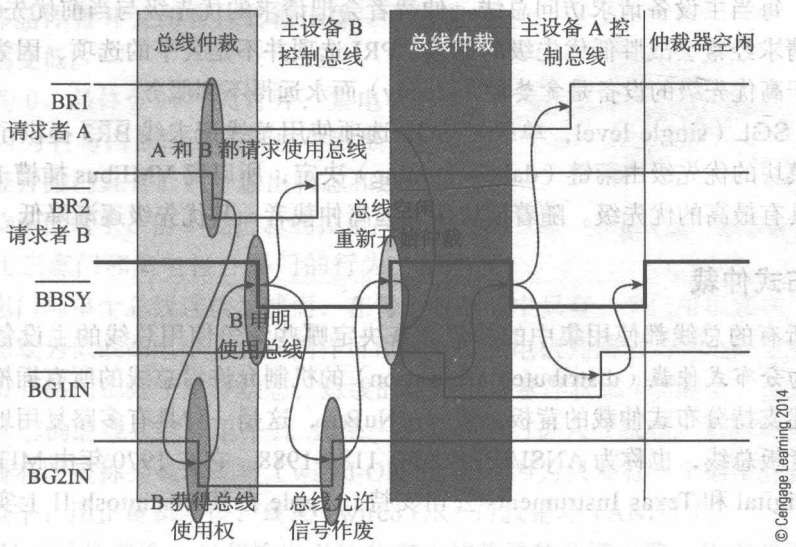


图 4-44 仲裁时序

当检测到已经发出 $\overline{BBSY}$ 信号后，仲裁者作废 $\overline{BG2IN}$ 信号。在该点，由于 $\overline{BBSY}$ 和总线允许信号是互锁的，因此 $\overline{BR2}$ 和 $\overline{BG2IN}$ 信号均为高电平无效，如图 4-44 所示。直到检测到 $\overline{BBSY}$ 信号为低电平，才允许仲裁者作废总线允许信号。当主设备 B 完成了其数据传输，请求者 B 发出 $\overline{BBSY}$ 信号。$\overline{BBSY}$ 信号作废的条件是保持 $\overline{BG2IN}$ 为高电平，此时从释放 $\overline{BR2}$ 信号开始至少延迟了 30ns。30ns 的延迟保证了仲裁者不会将旧的 $\overline{BR2}$ 信号解释为另一个请求。请求者 B 将等待至少 30ns 的间隔时间后再释放 $\overline{BBSY}$。

仲裁者检测到释放 $\overline{BBSY}$ 信号后将再次对总线请求进行仲裁。$\overline{BR1}$ 仍然是低电平有效，此时为唯一发出总线请求的信号线。仲裁者通过发出 $\overline{BG1IN}$ 信号允许请求者 A 使用总线。请求者 A 发出 $\overline{BBSY}$ 信号作为响应。当主设备 A 完成数据传输后，请求者 A 释放 $\overline{BBSY}$，前提是接收到 $\overline{BG1IN}$ 信号且从释放 $\overline{BR1}$ 信号开始至少延迟了 30ns。当请求者 A 释放 $\overline{BBSY}$ 后，由于没有总线请求，仲裁者空闲。

3. VMEbus 仲裁算法

插槽 1 中的仲裁者可以使用下列 3 种策略来处理带优先级的总线请求。请注意，这些策略是建议的策略，用户可以使用其他策略。

1）**选项 RRS**（round robin select，轮转选择） RRS 选项为每个主设备按照轮转的方式分配优先级。4 个级别的总线请求都有机会获得最高优先权。四级总线请求 $\overline{BR0}$ ～ $\overline{BR3}$，$\overline{BR3}$ 又接着 $\overline{BR0}$ 被看成一个循环；也就是说，连续的最高优先级的序列为 $\overline{BR0}$，$\overline{BR3}$，$\overline{BR2}$，$\overline{BR1}$，$\overline{BR0}$，$\overline{BR3}$，$\overline{BR2}$，……。在任意时刻，四级中的一个被赋予最高优先级，使得在此级别的请求者可能控制总线。如果当前具有最高优先级的请求者不想使用总线，下一级将被赋予最高优先级，以此类推。例如，当前最高优先级为 $\overline{BR2}$，下一次最高优先级将为 $\overline{BR1}$。

假设某个请求者被允许控制总线。当总线被释放后，下一级将被赋予新的最高优先级，该循环会一直进行下去。所有级别的总线请求依次成为最高优先级，不会落下一个。轮转选择仲裁是一种公平（fair）的方法，因为所有的主设备都享有平等的总线访问权。

2）**选项 PRI**（prioritized，优先级） PRI 选项为每个总线请求线分配优先级，$\overline{BR3}$（最高）～$\overline{BR0}$。每当主设备请求访问总线，仲裁者会把请求的优先级与当前优先级进行比较。高优先级的请求经常会战胜低优先级的请求。PRI 选项并不是公平的选项，因为低优先级的请求可能由于高优先级的设备是贪婪的（greedy）而永远得不到服务。

3）**选项 SGL**（single level，单级） SGL 选项使用总线请求线 $\overline{BR3}$ 提供了最小的仲裁机构。单个模块的优先级由菊链（daisy-chaining）决定，所以离 VMEbus 插槽 1 中的仲裁者最近的模块具有最高的优先级。随着模块位置远离仲裁者，其优先级逐渐降低。

4.6.2 分布式仲裁

并不是所有的总线都使用集中的仲裁者来决定哪些竞争使用总线的主设备可以控制总线。一种称为*分布式仲裁*（distributed arbitration）的机制允许沿总线的所有插槽依次进行仲裁。下面描述支持分布式仲裁的背板总线——NuBus，这是一种具有多路复用地址和数据线的通用同步背板总线，也称为 ANSI/IEEE STD 1186-1988。它在 1970 年由 MIT 构思，后来由 Western Digital 和 Texas Instruments 公司支持。Apple 在其 Macintosh II 上实现了 NuBus 的子集。

NuBus 系统中的每个卡都赋予了唯一的空间片段，它们共同构成了 Nubus 的整个地址空间；也就是说，NuBus 实现了*地理寻址*（graphic addressing）。32 位的 NuBus 地址可以用十六进制 $YXXXXXXX_{16}$ 表示，其中 Y 表示 15 个卡中的一个。$Y=1111_2$ 时，256MB 的地址空间范围从 $F0000000_{16}$ ～ $FFFFFFFF_{16}$ 被保留并称为*槽空间*（slot space）。该槽空间被分为 16 个 16MB 的块，每块与每个插槽相关联。因此，每个槽都有与之关联的唯一的 16MB 地址空间，并且，NuBus 不能支持超过 16 个插槽。

沿着背板的每个插槽都有自己唯一的*识别码*（ID），该识别码通过硬件连接到背板。每个槽的 ID 是固定的，由背板连接器本身而不是卡决定。例如，如果将卡插进插槽 11，低电平有效的背板线 $\overline{ID3}$ ～ $\overline{ID0}$ 与连接器相连以提供数值 0100（即 0100 取反后得到 1011，$1011_2=11_{10}$）。插槽标识线 $\overline{ID3}$ ～ $\overline{ID0}$ 并非沿着底板传输，只是简单地与地相连或与每个连接器的正极相连以提供恰当的插槽编号。

NuBus 仲裁

与 VMEbus 一样，NuBus 支持多处理和多个总线主设备。下面将解释 NuBus 是如何通过集电极开路门来实现分布式仲裁的。NuBus 有 4 个仲裁信号线 $\overline{ARB0}$ ～ $\overline{ARB3}$ 和一个优先级算法：当一组模块竞争使用总线时，具有最高 ID 号的模块获胜。未来的主设备通过发出其总线请求信号 $\overline{RQST}$ 开始仲裁。

NuBus 仲裁的关键是每个模块有唯一的插槽编号，从 0_{16} ～ F_{16}。当某个插槽中的卡需要总线仲裁时，该卡将其插槽编号放到总线上。不可思议的是，具有较低插槽编号的其他请求者将停止总线仲裁。同样，如果具有更高编号的插槽希望使用总线，正在请求的插槽将停止请求总线；也就是说，如果某卡需要总线仲裁，然后发现具有更高优先级的其他卡也需要总线仲裁，它就会*回避*（back off）。

为了理解分布式仲裁是如何工作的，读者需要理解*集电极开路*（open-collector）门。到目前为止，本书描述了两种门：传统的门，其输出不是 0 就是 1；三态门，其输出为 0、1 或悬浮（floating）。从历史上看，集电极开路门出现在三态门之前，用于允许多个设备来驱动相同的总线。

图 4-45 显示了一个集电极开路输出的反相器。该门的输出可以变为低电压。当门的输入为 1，内部晶体管开关闭合，其输出像一个正常的反向器被强制变低。

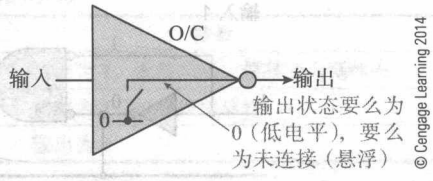


图 4-45 集电极开路门

当输入为 0，晶体管的开关打开，集电极开路门输出悬浮，因为它与内部的高或低电位线断开。也就是说，集电极开路门具有低电平输出状态和悬浮状态，可以把总线拉至低电平，但不能把总线拉至高电平。

下面对比三态门和集电极开路门的行为。用户可以将多个三态门与单个总线连接。然而，在每个时刻其中只有一个门可以使用，因为只有一个门可驱动总线为低或高电平。用户同样可以将多个集电极开路门与相一条总线连接。如果所有的集电极开路门都处于悬浮状态，总线的输出也是悬浮状态（实际上，通过上拉电阻可将处于悬浮状态的总线拉到高电平）。如果一个或多个门进入 0 状态，总线将被迫进入 0 状态。这种安排有时被称为线或逻辑（wired-OR logic），因为只要有一个输出为低电平，整个输出就为低电平；用正逻辑术语，线或（wired OR）门就是与（AND）门。

图 4-46 显示了具有输入 X 和输出 Y 的分布式仲裁器的关键电路。该电路也与总线上的一条仲裁控制线相连。接下来关注的是电路和总线状态之间的关系。首先看一下电路输入与输出之间的基本逻辑传递函数。仔细分析该电路的逻辑，会发现输出为：

$$Y=\overline{P}\cdot\overline{Q}$$

由于 $P=X$ 且 $Q=\overline{X}$，因此

$$Y=\overline{P}\cdot\overline{Q}=\overline{X}\cdot\overline{\overline{X}}=\overline{X}\cdot X=0$$

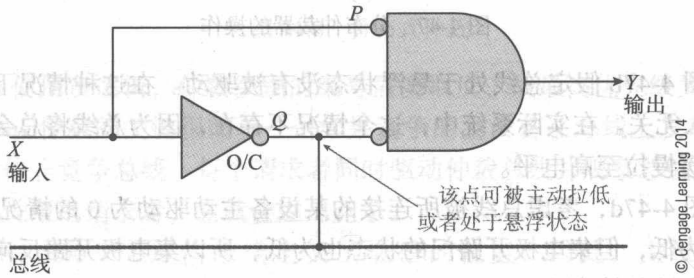


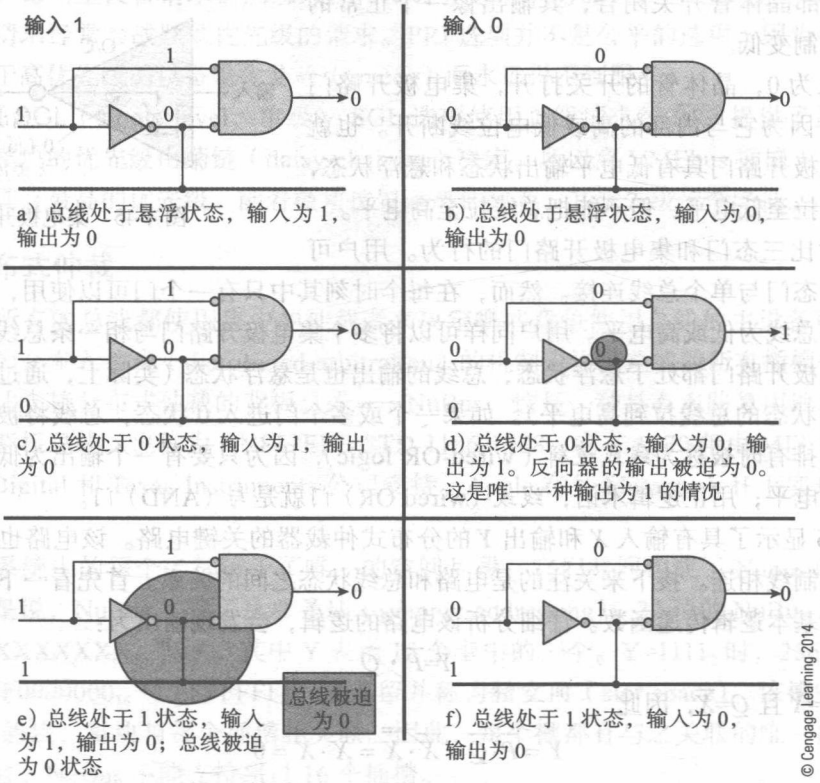
图 4-46 分布仲裁机制

乍一看，该电路似乎什么也没有做，因为无论其输入是什么，其输出总是 0。但是，因为集电极开路反相器的输出与总线相连，如果 X=1（即 Q=0）该电路将驱动总线为 0。如果 X 的输入为 0，集电极开路门将处于悬浮状态，总线不会被驱动。默认情况下，总线将通过电阻被拉至高电平状态。

这就是其中聪明的地方。假设 X 的输入为 0，总线为低电平，因为另一个设备将其驱动为低电平。在这种情况下，集电极开路门反相器的输出也被总线强制为低。此时，与门的两个输入端都为 0，Y 的输出将为 1。也就是说，Y 的输出为 0，除非 X 的输入为 1 且总线被驱动为低。这样就得到了一种机制，它可以主动地将总线驱动为低，或者在需要将总线驱动为高时检测是否有另一个设备将驱动总线为低。该机制成为分布式仲裁的基础。

图 4-47 显示了图 4-46 中分布式仲裁器考虑所有可能的输入条件和总线的状态所进行的操作。记住，总线可以为悬浮状态（没有被驱动）或者主动下拉至低电平。当其处于悬浮状

态时，电阻将缓慢将总线拉至高电平。



a）总线处于悬浮状态，输入为 1，输出为 0

b）总线处于悬浮状态，输入为 0，输出为 0

c）总线处于 0 状态，输入为 1，输出为 0

d）总线处于 0 状态，输入为 0，输出为 1。反向器的输出被迫为 0。这是唯一一种输出为 1 的情况

e）总线处于 1 状态，输入为 1，输出为 0；总线被迫为 0 状态

f）总线处于 1 状态，输入为 0，输出为 0

图 4-47 分布仲裁器的操作

图 4-47a 和图 4-47b 假定总线处于悬浮状态没有被驱动。在这种情况下，电路的输出总是 0，与它的输入无关。在实际系统中，这个情况不存在，因为总线将总会被主动拉低至低电平或者被电阻缓慢拉至高电平。

图 4-47c 和图 4-47d，考虑总线被所连接的某设备主动驱动为 0 的情况。在图 4-47c 中，总线被主动驱动为低，但集电极开路门的状态也为低，所以集电极开路反向器的输出和总线之间没有冲突。电路的输出为 0。

在图 4-47d 中，输入为 0，集电极开路门的输出悬浮。然而，总线的状态为低，反相器的输出被拉低至低电平。电路的输出现在为 1。该输出告诉系统，另一个设备将总线驱动为低，与输入矛盾。

图 4-47e 和图 4-47f，因为没有其他设备将总线驱动为低，总线处于高状态。图 4-47e 是一个有趣的例子。其输入为 1，集电极开路门的输出为低电平。它将驱动总线为低状态。此时，该电路驱动总线，电路的输出为 0。在图 4-47f 中，输入为 0 且反相器的输出为悬浮状态，所以与总线的状态没有冲突。

正如读者可以看到的，有两种特殊的情况。一种情况是，总线为低电平有效，但反相器的输出为高，这导致反相器的输出被拉低。另一种情况是，总线为高状态，反相器的输入为低电平有效，导致总线被迫变为低状态。

表 4-4 总结了该电路的动作。电路的输入代表“想要使用总线（I want the bus）”或“不想使用总线（I don’t want the bus）”状态。如果总线没有被驱动为低，当电路输入为 1 时，该

电路将驱动总线为低。该电路产生 0 输出，除非其输入为 1 且总线被其他设备主动驱动为低。

表 4-4 图 4-47 中总线状态的小结

条件	总线状态	结果
想要使用总线	总线空闲（高电平）	输出为 1。获得总线，使其变为低状态
想要使用总线	总线忙（低电平）	输出为 1。不想使用总线
不想使用总线	无所谓	输出为 0

图 4-48 显示了 NuBus 仲裁器的细节，该仲裁器使用图 4-46 中电路的原理。潜在的需要使用总线的主设备将其仲裁级别放至 4 位仲裁总线 $\overline{ID3}$ ～ $\overline{ID0}$ 上。注意，图中只画出了仲裁器 4 个阶段中的 3 个阶段。

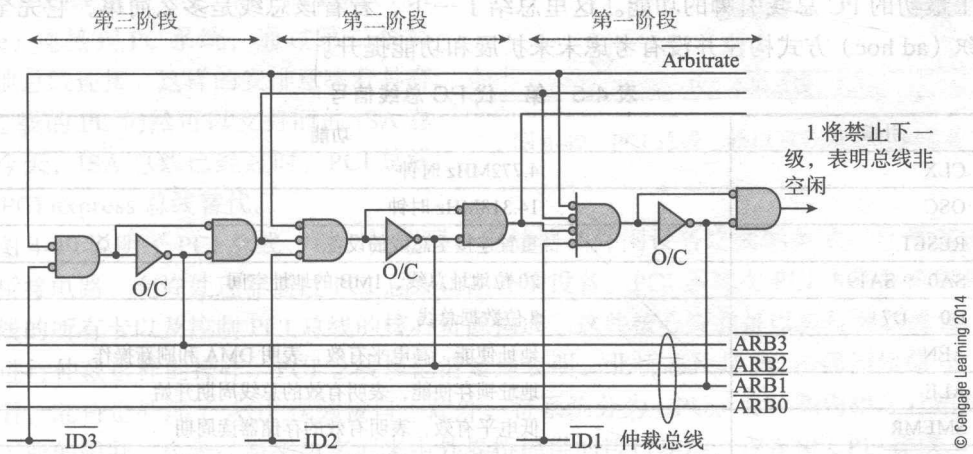


图 4-48 分布式仲裁

由于 NuBus 使用负逻辑，仲裁级别的编号需要取反，故最高优先级为 0000 而最低优先级为 1111。NuBus 仲裁的关键很简单——如果竞争的主设备在总线上发现具有比自己更高的优先级，它就停止竞争总线。每个请求者同时驱动仲裁总线并观察总线上的信号。如果它发现具有更高优先级的请求者，它就会退出竞争。

在图 4-48 中，$\overline{ID0}$ ～ $\overline{ID3}$ 信号线定义了插槽的位置，因此，主设备的优先级以及 $\overline{ARB0}$ ～ $\overline{ARB3}$ 信号都是仲裁信号线并沿着总线传输。标志为 Arbitrate 的信号允许主设备开始总线仲裁，如果主设备赢得仲裁将输出 $\overline{GRANT}$ 信号。NuBus 使用非顺序的仲裁逻辑，仲裁在总线正常活动时并行发生。

假设三个主设备的级别编号为 0100（4）、0101（5）和 0010（2），它们同时把代码 1011、1010 和 1101 分别放到仲裁总线上。由于仲裁线是集电极开路的，任何为 0 级的输出将使总线拉低为 0。在这个例子中，总线将被迫进入 1000 状态。在第二阶段的主设备把代码 1101 放到仲裁总线上，将检测到 $\overline{ARB2}$ 被拉低，结束仲裁过程。仲裁总线当前的值为 1010。具有代码 1011 的主设备检测 $\overline{ARB0}$ 被拉低，离开仲裁过程。仲裁总线当前的值为 1010，具有该值的主设备获得总线的控制权。

由于 NuBus 实现了优先级仲裁系统，高优先级的插槽可以独占总线并阻止低优先级的插槽获得控制。这种总线占用冲突可以被延迟（deferral）消除。一旦插槽获得了总线控制权，然后放弃它，该插槽将不会试图重新获得总线控制权，直到所有等待的总线请求已经被处理。NuBus 确实具有允许总线主设备维持其总线控制权的特殊机制。称为关注周期

（attention cycle）的特殊 NuBus 控制周期，可以用来请求继续拥有总线的所有权。

上面介绍了背板总线并描述了分布式（distributed）和本地化（localized）仲裁协议的例子。事实上，在总线设计中有很多的变种，几乎每一个工程师都梦想着创建一种终极总线，就像所有的作者都想写“伟大的美国小说（The Great American Novel）”一样。下面将介绍更为复杂的 PC 总线——PCI 总线，及其衍生物 PCI express（PCIe）。

4.7 PCI 和 PCIe 总线

在介绍 PCI 总线之前，需要了解其应用环境。最初的 PC 总线使用 62 个插头的连接器，支持 20 位地址总线，工作频率为 4.772MHz。总线其实就是一种 CPU 接口的扩展。表 4-5 描述了最初的 PC 总线引脚的功能。这里总结了一下，看看该总线是多么简单，它完全是以自组织（ad hoc）方式构建并没有考虑未来扩展和功能提升。

表 4-5 第一代 PC 总线信号

引脚	功能
CLX	4.772MHz 时钟
OSC	14.318MHz 时钟
RESET	重置连接至总线的设备
SA0 ～ SA19	20 位地址总线，1MB 的地址空间
D0 ～ D7	8 位数据总线
AEN	地址使能，高电平有效，表明 DMA 和刷新操作
ALE	地址锁存使能，表明有效的总线周期开始
$\overline{\text{SMEMR}}$	低电平有效，表明有效的存储器读周期
$\overline{\text{SMEMW}}$	高电平有效，表明有效的存储器写周期
$\overline{\text{IOR}}$	低电平有效，表明有效的 I/O 端口读周期
$\overline{\text{IOW}}$	高电平有效，表明有效的 I/O 端口写周期
$\overline{\text{IOCHRDY}}$	由总线从设备使用来扩展总线周期
$\overline{\text{OWS}}$	零等待状态，用来表明没有等待状态
$\overline{\text{IOCJCHK}}$	奇偶校验状态，用来表明存储器错误
DRQ1 ～ DRQ3	外围设备使用的 DMA 请求线
$\overline{\text{DACK1}}$ ～ $\overline{\text{DACK3}}$	CPU 发给外围设备的 DMA 允许信号线
T/C	指示 DMA 操作的结束
$\overline{\text{REF}}$	刷新信号，表明 DRAM 开始执行刷新周期
IRQ2 ～ IRQ7	中断请求线

4.7.1 PCI 总线

外围组件互连局部总线（Peripheral Component Interconnect local bus，PCI bus）代表了一种对 PC 系统架构的激进变革。Intel 在 1993 年底设计了这种用于基于 Pentium 系统的总线。PCI 总线不仅比以前的总线快得多；它还大大扩展了 PC 体系结构的功能。事实上，PCI 总线是 PC 扩展性和灵活性的核心。PCI 允许用户把卡插到计算机系统中，通过增加调制解调器、SCSI 接口、视频处理器、声卡等模块来增加功能。更具体地说，PCI 总线允许这些卡利用过众所周知的北桥（North bridge）与 CPU 通信。总线接口电路总称为芯片组（chipset）。所有与 PCI 相连的 PC 都需要这样的一个芯片组。

与来自 CPU 的地址、数据和控制信号相比，PCI 被称为本地总线（local bus）或称为局

部总线。将系统与 CPU 直接连接可提供最快的数据传输率，与 CPU 直接连接的总线被称为前端总线（front side bus）。

PCI 总线支持即插即用（plug-and-play）功能，插入的 PCI 卡在加电时自动配置，一些类似中断请求的资源被分配给即插即用卡，该过程对用户来说是透明的。最初的 PCI 总线工作在 33MHz，支持 32 位和 64 位数据总线。PCI 总线 2.1 版本支持 66MHz 的时钟。

PCI 总线通过单芯片的 PCI 桥（PCI bridge）连接到 PC 系统，通过第二个桥与其他总线连接。这样的安排意味着具有 PCI 总线的 PC 仍然可以支持旧的 ISA 总线。今天，ISA 总线已经过时，PCI 总线也由 PCI express 总线替代。

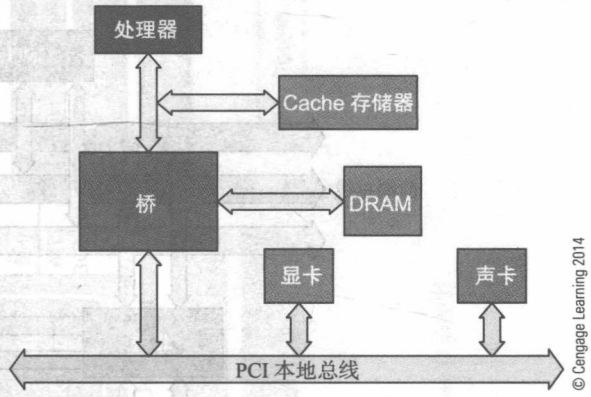


图 4-49　PCI 总线、桥以及处理器之间的关系

图 4-49 说明了 PCI 总线、桥、处理器、存储器和外围设备之间的关系。处理器直接连接到桥接电路，允许处理器通过 PCI 总线访问外围设备。PCI 系统由 PCI 本地总线本身、插入总线的所有卡以及控制 PCI 总线的核心资源构成。这些核心资源可以执行例如对插入总线的卡进行仲裁这样的操作。PCI 总线规范清楚地表明，北桥芯片是 PC 的强制性部分。从历史上看，北桥也处理了 AGP 视频接口。此外，将系统分为 CPU、北桥和南桥芯片组是由于制造工业的限制。今天，越来越多原来由外界桥提供的接口功能，现在由 CPU 提供。

PCI 连接器系列

根据插槽的宽度（32 位或 64 位）以及电压水平（5V 电路或更现代的 3.3V 电路）可将 PCI 连接器分为几个版本。当然，将 PCI 卡插入错误类型的插槽中可能损坏卡或主板。为了确保卡被正确插入（正确的方式和正确的电压版本），主板上的每个 PCI 插座在连接器间都有一个或多个凸起（key）。除非有与凸起对应的缺口，卡不能插入插座，如下图所示。

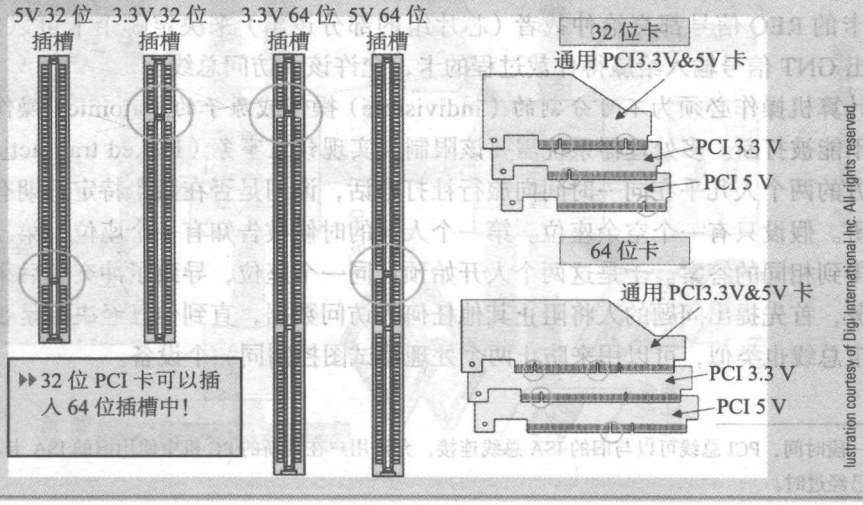


图 4-50 显示了一个具有 PCI 本地总线和 ISA 总线[⊖]的 PC 的系统框图。第二个桥，通常被称为南桥（south bridge），将 PCI 和 ISA 总线连接起来。图 4-51a 说明了 Pentium 4、Intel 芯片组以及 PCI 总线之间的关系。图 4-51b 说明了更加现代的 Intel Core i7 处理器接口。

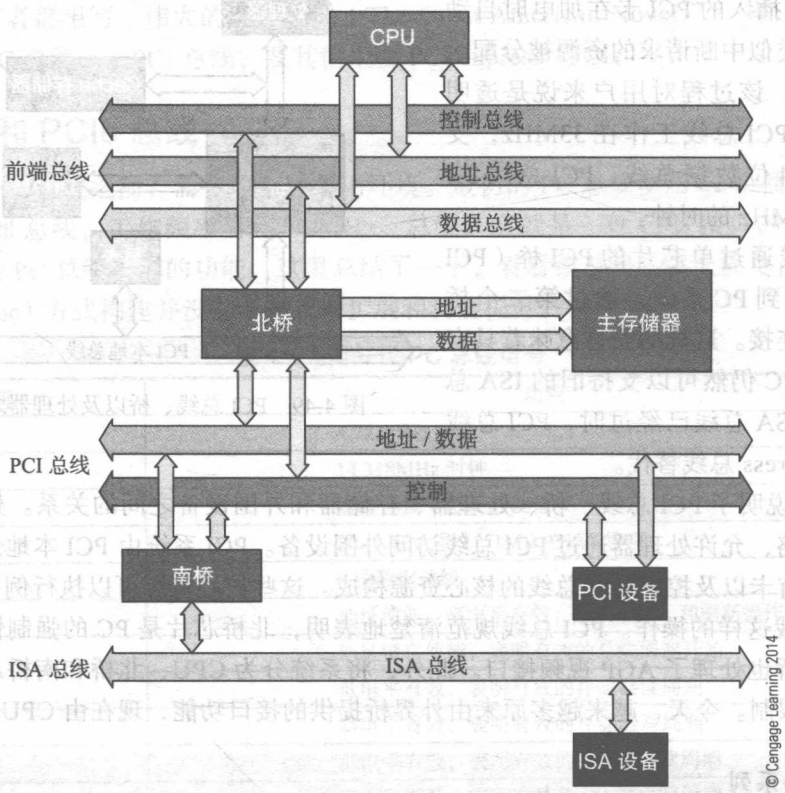


图 4-50 北桥、南桥和 PCI 总线

每个插入 PCI 总线的卡可能包含总线主设备，能够通过控制 PCI 总线来控制数据的流动。每个卡或具有总线主设备功能的代理（agent）需要向 PCI 总线提供两个信号：REQ（请求总线）和 GNT（从总线接收到允许）。核心资源（即北桥）负责处理 PCI 总线仲裁和处理来自每个代理的 REQ/GNT 信号对。当某个卡上的设备希望使用总线，它发出其 REQ 信号。所有 PCI 卡的 REQ 信号都交给仲裁者（芯片组的部分逻辑）来决定哪个卡可以使用总线。仲裁者发出 GNT 信号输入给赢得仲裁过程的卡，允许该卡访问总线。

某些计算机操作必须为不可分割的（indivisible）操作或原子的（atomic）操作；也就是说，操作不能被打断。多处理器系统需要该限制来实现锁定事务（locked transaction）。假设在不同地方的两个人几乎在同一时间向旅行社打电话，询问是否在某个特定日期有从伦敦到巴黎的航班。假设只有一个空余座位。第一个人问的时候被告知有一个座位。第二个人问的时候也会得到相同的答案。于是这两个人开始预订同一个座位，导致了冲突的后果。通过使用锁定事务，首先提出问题的人将阻止其他任何人访问数据，直到他已经决定是否要预订该座位。锁定总线也类似，可以用来防止两个处理器试图控制同一个设备。

⊖ 有一段时间，PCI 总线可以与旧的 ISA 总线连接，允许用户在其新的 PC 机中使用旧的 ISA 卡。ISA 总线现在已经过时。

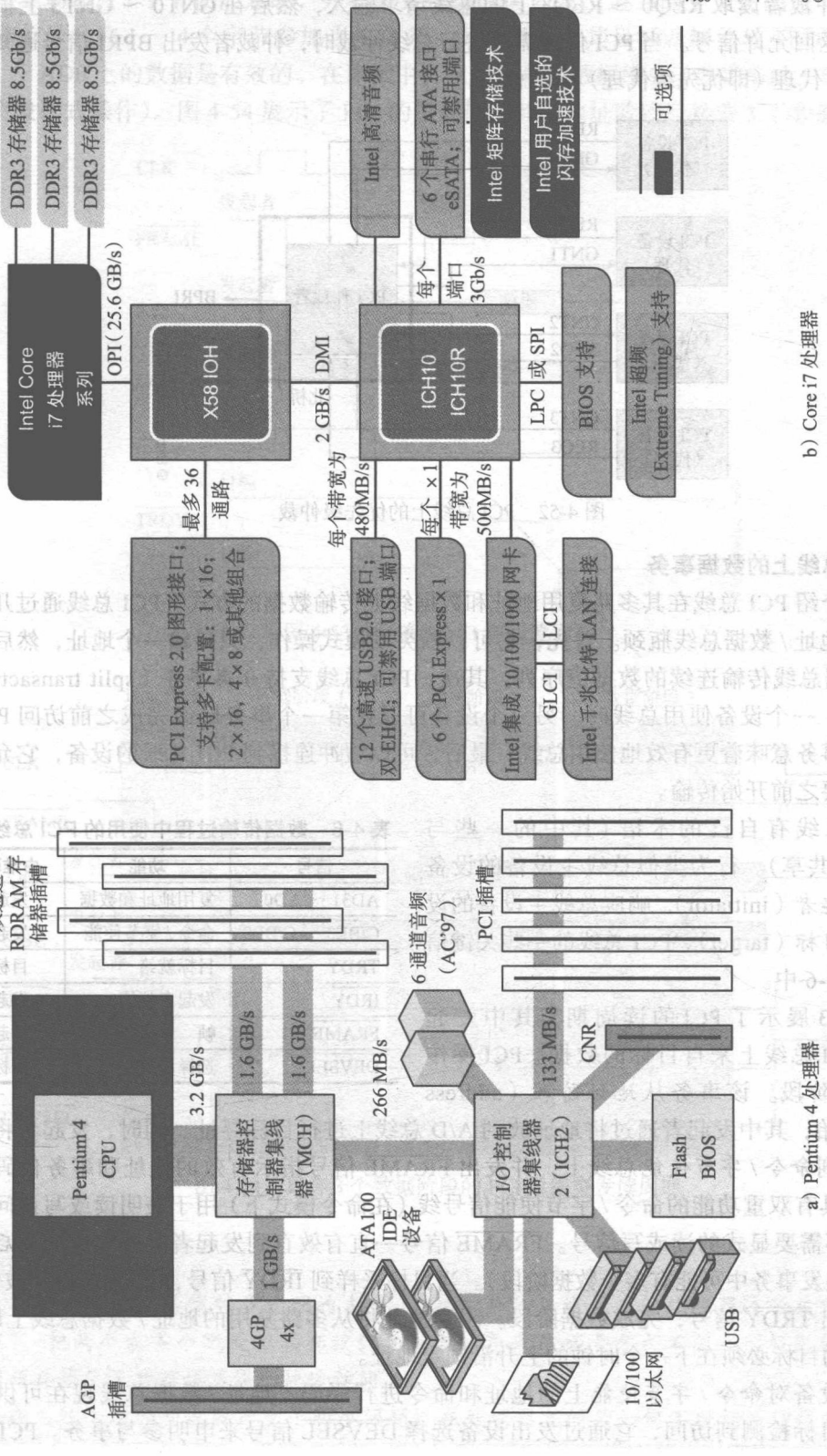


图 4-51 IA32 芯片组

图 4-52 显示了如何实现 PCI 总线仲裁。REQ 和 GNT 信号连接至组成北桥一部分的仲裁者。该仲裁者读取 REQ0 ～ REQ3 上的总线请求输入，然后在 GNT0 ～ GNT3 上根据仲裁获胜者返回允许信号。当 PCI 代理需要进行总线仲裁时，仲裁者发出 BPRI 信号通知主处理器，PCI 代理（即优先级代理）需要总线。

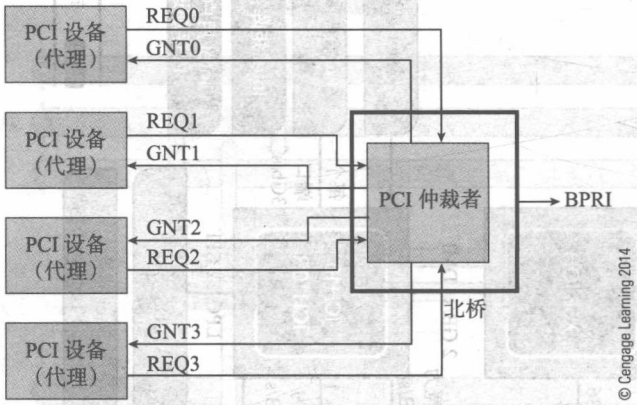


图 4-52 PCI 总线上的优先级仲裁

PCI 总线上的数据事务

下面介绍 PCI 总线在其多路复用地址和数据线上传输数据的方式。PCI 总线通过几种方法来补偿地址 / 数据总线瓶颈。首先，它可以以突发模式操作，只传输一个地址，然后使用地址 / 数据总线传输连续的数据值序列。其次，PCI 总线支持分离事务（split transaction）；也就是说，一个设备使用总线时，另一个设备可以在第一个事务已经完成之前访问 PCI 总线。分离事务意味着更有效地使用总线。最后，可以缓冲连接到 PCI 总线的设备，它允许数据在被需要之前开始传输。

PCI 总线有自己的术语（其中的一些与 SCSI 系统共享）。行为类似总线主设备的设备被称为发起者（initiator），响应总线主设备的设备被称为目标（target）。PCI 总线的一些关键信号列在表 4-6 中。

表 4-6 数据传输过程中使用的 PCI 总线信号

信号	功能	由谁驱动
AD31 ～ AD0	复用地址和数据	发起者
$\overline{C/BE3}$ ～ $\overline{C/BE0}$	命令 / 字节使能	发起者
$\overline{TRDY}$	目标就绪	目标
$\overline{IRDY}$	发起者就绪	发起者
$\overline{FRAME}$	帧	发起者
$\overline{DEVSEL}$	选择设备	目标

图 4-53 展示了 PCI 的读周期，其中发起者读取 PCI 总线上来自目标的数据。PCI 操作分为几个阶段。该事务从地址阶段（address phase）开始，其中发起者通过将地址放到 A/D 总线上进行目标寻址。同时，发起者将 4 位的代码放到命令 / 字节使能总线上，并发出 $\overline{FRAME}$ 信号指示有效的地址和事务代码。因为使用了具有双重功能的命令 / 字节使能信号线（在命令模式下）用于表明读或写访问，故 PCI 总线不需要显式的读或写信号。$\overline{FRAME}$ 信号一直有效直到发起者准备完成的最后数据阶段（在突发事务中可能有多个数据阶段）。当目标采样到 $\overline{IRDY}$ 信号，$\overline{FRAME}$ 信号被作废，且目标发出 $\overline{TRDY}$ 信号，完成数据阶段。因为地址将从多路复用的地址 / 数据总线上删除，所有潜在的目标必须在下一个时钟的上升沿锁定地址。

目标设备对命令 / 字节使能上的地址和命令进行译码。地址 / 数据总线现在可供目标使用。当目标检测到访问，它通过发出设备选择 $\overline{DEVSEL}$ 信号来申明参与事务。PCI 协议

规定目标必须在预定的时间内申请使用总线，否则该事务将被终止。发起者使用当前命令值停止驱动命令/字节使能总线，并在事务持续时间内发出适当的字节使能信号。这些字节使能信号允许1～4个字节参加当前事务。发起者发出其目标就绪信号$\overline{\text{TRDY}}$，表明在AD31～AD0上的数据是有效的。在该例中，发生了单个数据事务（也就是说，当前事务并不是突发模式操作）。图4-54展示了PCI的读周期，其中地址阶段后接着3个数据阶段。

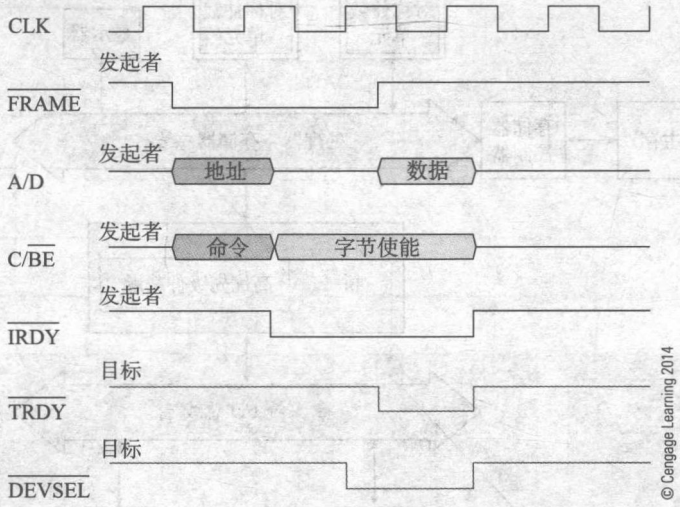


图4-53 简单的PCI总线读周期

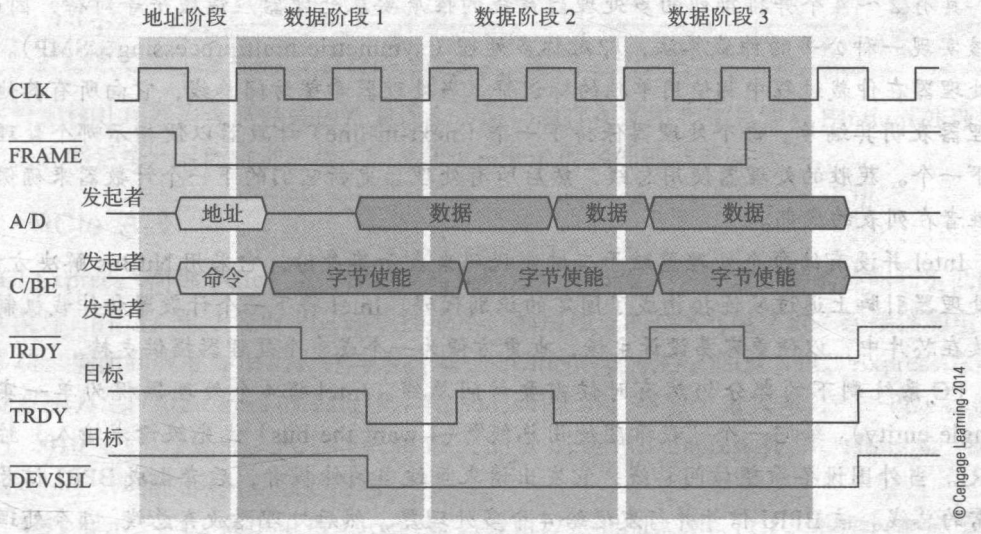


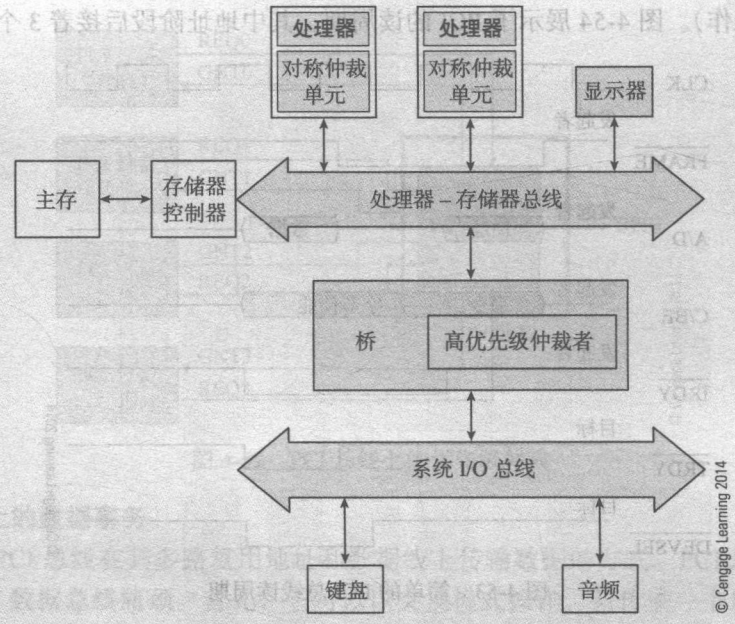
图4-54 具有3个数据阶段的PCI总线突发读周期

多处理和PCI总线

本节介绍PCI总线对多处理的支持（注意，PCI总线在多核处理器流行之前就已经出现）。把两个或多个微处理器都放到主板上，与采用插卡方式不同，这样做解决了散热问题并简化了对前端总线的快速访问。

在主板上增加处理器产生了两个问题：在处理器之间是如何实现相互仲裁的？系统

其余部分的需求是什么？记住，系统的其余部分包括设备和子系统，它们也需要访问系统总线以实现DMA等目的。下图说明了一个基于PC的多处理器系统，它的两个处理器与处理器－存储器总线连接，I/O总线和处理器总线之间有一个桥。

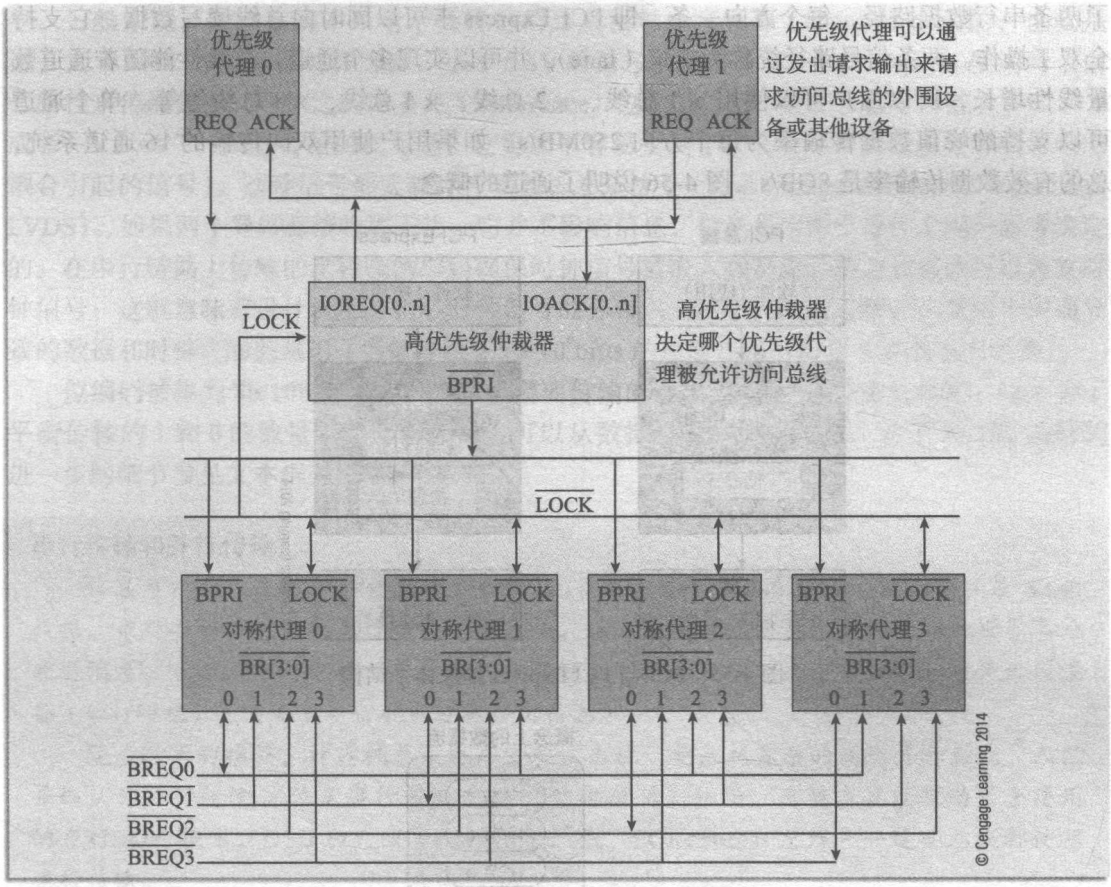


具有2～4个并排排列的多处理器系统的性质要求处理器应该被平等对待，因此，应该实现一种公平的仲裁算法，即对称多处理（symmetric multiprocessing，SMP）。每个处理器在仲裁过程中都使用单独的标识符。当处理器希望访问总线，它向所有其他的处理器表明其编号。每个处理器保持下一个（next-in-line）计数器以便指示哪个处理器是下一个。获胜的处理器使用总线，然后所有处理器更新它们的下一个计数器来确保该获胜者在列表的底部。

Intel并没有给每个处理器赋予二进制代码来表示其身份，它采用NuBus解决方案，在处理器引脚上通过硬连接构成了固定的识别代码。Intel将下一个计数器和仲裁机制一起放在芯片中，以便更容易设计主板，也更方便为一个或多个处理器提供支持。

PC系统剩下的部分仍然有时候需要访问总线。Intel将4个处理器视为单一实体（single entity），给它一个“我希望使用总线”（I want the bus）优先级请求输入，称为BPRI。当外围设备希望访问总线，它发出请求至适当的仲裁者，后者生成BPRI信号至所需的总线。该BPRI信号并行发送给4个多处理器，然后处理器放弃总线。4个处理器一起完成原来由一个处理器单独完成的工作。

一旦处理器之一希望使用总线，它与其他处理器一起仲裁获得总线（必须服从SMP仲裁机制的公平性约束）。当优先级代理希望使用总线，该代理从一堆处理器那里得到总线就像与单个处理器打交道一样。下图演示了对称多处理和优先级仲裁之间的关系，其中一个设备而不是一个处理器请求访问总线。



尽管多核处理器当今已经成为普遍现象，但仍然存在采用 PCI 总线与 Pentium 处理器来实现多处理器系统。后面将进一步介绍基于 Pentium 的多处理器系统。

4.7.2 PCIe 总线

PCI Express（简称 PCIe）总线针对高科技应用环境开发，它要求更高的性能和更低的成本。其设计目标是：低于现有 PCI 总线的成本，利用现成的技术（电路板、连接器和电路），支持移动、桌面和服务器市场，并与现有基于 PCI 的系统兼容。因此，PCI Express 总线的实现没有压力。PCI Express 的 1.0a 版本在 2003 年推出，第三个重大修订——PCI Express 3.0，于 2010 年 11 月发布。PCI 和 PCIe 最大的区别是 PCI Express 使用*串行传输*（serial transmission），即实现点对点数据传输。

图 4-55 展示了 PCI 总线和 PCI Express 总线协议之间的区别。PCI 总线协议响应了 ISO（国际标准化组织）的开放系统互连（open system interconnection，OSI）模型，它试图将所有通信系统划分为 7 个抽象层，每一层为其上层执行特定的功能。层次化协议的优势是任意一层都可以在不影响其上层和下层的基础上被新技术替换。例如，物理层可以从 PCB 上的铜线变为光纤而不需要改变数据链路层或事务层。

PCI Express 协议的最低层是*物理层*（physical layer），负责点对点数据位的传输。与传统的 PCI 并行数据 / 地址总线不同，PCI Express 使用串行总线，其中数据沿着一条或者一对使用差分编码的信号线一位一位地传输。PCI Express 的一个特别有趣的方面是，它提供

了两条串行数据路径，每个方向一条，即 PCI Express 卡可以同时向总线读写数据，它支持全双工操作。两条信号路径统称为通道（lane），并可以实现多个通道。总线性能随着通道数量线性增长，所以用户可以使用 ×1 总线、×2 总线、×4 总线、×8 总线等等。单个通道可以支持的峰值数据传输率为每个方向 250MB/s。如果用户使用双向传输的 16 通道系统，总的有效数据传输率是 8GB/s。图 4-56 说明了通道的概念。

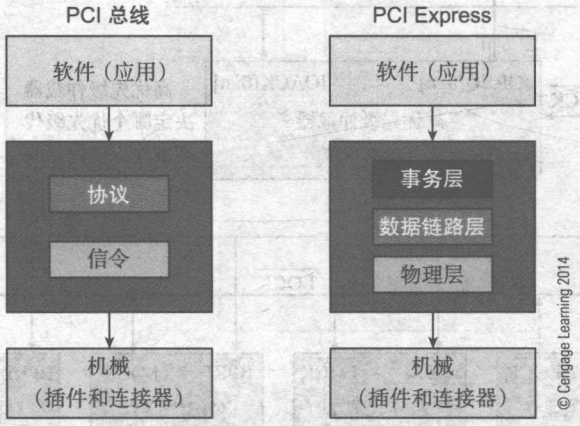


图 4-55 PCI 与 PCI Express 抽象体系结构

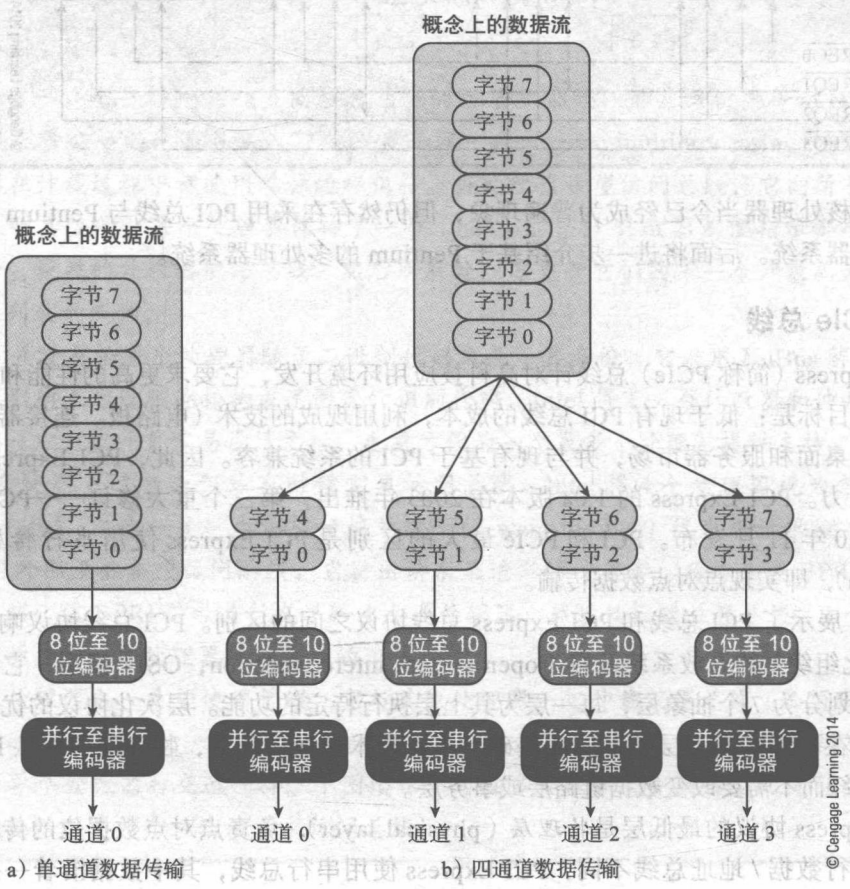


图 4-56 PCI Express 物理层传输和通道

一般来说，数字系统中以电平描述的信息是相对地或底部（chassis）来说的；也就是说，大于3.0V的信号被解释为高，小于0.3V的信号被解释为低。PCI Express使用两个信号路径来传输数据，导体之间的差异（difference）包含了信息；例如，信号可能为+V、−V或者−V、+V[⊖]。差动传输的优点是，它更加不容易受到干扰（噪声和其他由电容或者电感耦合引起的信号）。这种信号形式被称为低电压差分信号（low voltage differential signaling，LVDS）。如果两个导体都接收到干扰，它并不影响信息，信息是由两个导体之间的差异决定的。在串行链路上传输的比特流的编码确保时钟信号被嵌入数据流，使用数据流可以恢复时钟信号；这就意味着设计者不必担心时钟信号的分布以及由于信号通过路径长度的不同而导致的数据和时钟之间的延迟（当信号以2.5×10^9bit/s的速度传输时需要考虑的重要因素）。

位编码被称为8b/10b编码是因为每个需要传输的8位数据以10位进行传输，这是为了平衡传输的1和0的数量来保证时钟信号可以从数据信号中恢复过来。关于8b/10b编码的进一步的细节参见文本框“8b/10b编码”。

串行传输和并行传输

信息可以逐位进行串行传输或并行传输。64位总线可以在两点之间同时传输64位数据。串行数据传输系统的原型是电报系统，该系统设计用来使用摩斯电码向世界各地发送消息。后来，串行电报让位于串行电话连接。计算机基本上在处理器和慢速外围设备（如打印机）之间使用串行数据连接传输信息。

随着时间的推移，计算机总线也在发展，出现了越来越复杂的高速并行总线。人们普遍认为并行传输m位是串行传输m位速度的m倍。然而，目前在硬盘驱动器上使用的串行ATA技术，USB和火线（firewire）总线，PCI express总线——这些总线都使用串行传输。

当然，因为总线的设计和连接器的成本，并行传输比串行传输昂贵得多。然而，在数据传输中还涉及其他更微妙的因素。首先，总线连线的工作类似天线，既在数据沿着总线传输时辐射信息，也接收来自相邻总线连线上的信息（干扰）。在速度较高的情况下，电磁干扰问题变得十分严重——并行总线中并行传输的导线越多这种情况就越严重。串行总线只需要两条传输线，这使得它更容易免受干扰的影响。

高速并行总线遇到的另一个问题是数据扭曲（data skew）。将32位数据交给32条数据线，这些数据沿着总线传输给32个接收器，由于总线驱动器、信号传播路径和接收器上不同的延迟，这32位数据并不是在完全相同的时间到达目的地。接收信号的时间范围就是数据扭曲。过去，数据扭曲并不是一个重要的问题，因为信号（时钟）频率很低。今天，随着信号频率大于1GHz，数据扭曲成为一个大问题。需要复杂的设计技术和抗扭曲电路来处理该问题。串行数据不会出现这个问题，数据必须按照传送的顺序接收。当然，在数据和时钟（需要时钟将数据流分解成单个比特）之间也可能出现扭曲。然而，许多串行数据传输系统实现串行数据编码，这使时钟信号可以从数据本身中恢复。

并行数据系统本质上是半双工的，这意味着每个时刻只允许在一个方向上从一点到另一点传播数据。当然，可以反向传输，但逆转发送者和接收者的角色需要时间。虽然

⊖ LVDS数据路径使用电流开关。+3.5mA或−3.5mA的电流从发送端注入总线。在总线的远端，有一个100Ω的终端电阻。根据欧姆定律$E=IR$，导线的两端电压为350mV。

串行通道也是半双工的，它很容易提供两个串行通道，每个方向一个，但实现全双工传输可以同时完成两个方向的传输。

有意思的是，PCI express 总线为串并行提供了最好解决方案。它提供了一对串行通道，可以以非常高的信号速度实现全双工功能。此外，它还允许使用多个通道（每个通道由两个串行通道组成）来实现有效的并行总线。

最后，并行总线数据传输协议和辅助功能（例如总线仲裁和中断处理）需要复杂的控制结构。串行总线被迫采取一刀切的策略，所有的功能都以串行信息传输而不需要额外的控制线路。

PCIe 数据链路层

在详细介绍数据链路层之前，有必要描述 PCIe 总线（或通过其他具有分层协议的串行总线，包括 Internet 本身）上传输数据的格式。在支持分层协议的系统上传输的数据看上去有点像俄罗斯套娃的多层封装。在链路的一端，应用程序发出数据包并用分隔符包装。然后，应用层将数据包交给另一层（如，数据链路层），该层然后再用自己的终止符封装数据。数据链路层将数据传递给物理层，在该层也增加了开始和结束标志。当传输数据时，消息的两端具有 3 个标志——每层添加一个标志。图 4-57 展示了使用三层或三级系统封装的概念图。每个协议层为传递给下一层的信息添加一个头部和尾部。同样，每一层将传递给上一层的消息去除头部和尾部。

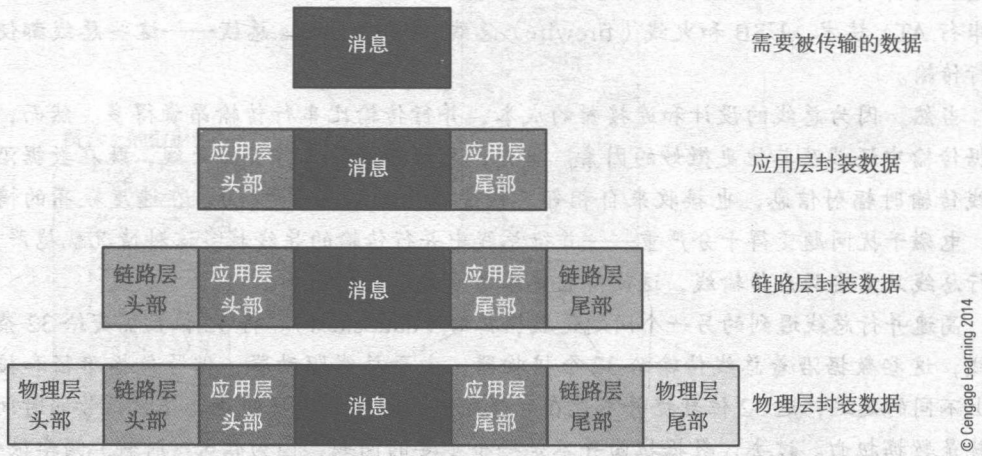


图 4-57 消息封装

图 4-58 展示了 PCIe 总线的消息结构。最高层为事务层（transaction layer），由头部和实际消息本身组成。头部定义了数据消息的性质，例如数据的地址信息（后面将更详细地讨论头部的内容）。事务层的尾部为错误检测码（error-detecting code，ECRC）。事务层提供 4 个地址空间：存储器、I/O、配置和消息空间（message space）。消息空间这个术语指的是执行控制功能的消息集合，即总线操作被指定为命令的编码（就像 CPU 的操作码），而不采用专门的控制线路。消息空间用来支持中断和复位（以及所有其他形式的硬件管理）。通过使用消息空间，像 VMEbus 或者 NuBus 那样的总线物理控制线不是必要的。

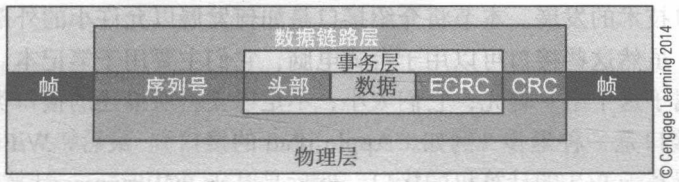


图 4-58 PCIe 总线的串行数据结构

8b/10b 编码

8b/10b 编码是一种使用 10 位携带 8 位信息进行串行数据传输的方式，是相对现代的数据编码/解码机制（由 IBM 的 Widmer 和 Franaszek 在 1983 年提出）。每个字节需要额外的两位（或者说冗余 25%）从而提高了传输机制的性能。10 位代码限制包括 5 个 1 和 5 个 0，或者 4 个 1 和 6 个 0，或者 6 个 1 和 4 个 0。这样可以确保不会出现一长串的 1 或者 0。此外，使用称为运行偏差（Running Disparity）的机制，以确保平均具有相同数量的 1 和 0；这对于信号中没有直流分量的情形是必要的。

8b/10b 编码算法通过一个 8 位的字节来进行操作，将 8 位分解为 3 位和 5 位两组。这两组分别进行编码，5 位的组成为 6 位的编码，3 位的组成为 4 位的编码。这两个编码通过重新排列合并成一个 10 位的编码。注意，除了该算法生成的 256 个有效代码外，还有 12 个代码被保留以实现控制功能。

数据链路层添加的头部包括来自事务层的帧的序号，并附加一个循环冗余校验（cyclic redundancy check，CRC）用来保证数据链路层的错误检测；也就是说，如果数据链路层的帧上的任何信息被损坏，该错误会被检测出来并要求重传。数据链路层只有在知道有一个可用于接收报文的缓冲区时才发送报文，这样就可以避免因丢失包而重传（这是早期串行数据链路协议，如 HDLC，的共同特征）。

PCIe 总线上数据交换的规范是比较复杂的。图 4-59 给出了包含 12 或 16 字节的数据包报头的一般结构。正如读者所看到的，它包含 11 个字段（两个留给将来使用）。该结构意味与传统总线相关的所有硬件开销（仲裁、中断、握手，等等）变为冗余的，代价是增加了延迟和由于数据开销带来的效率降低。

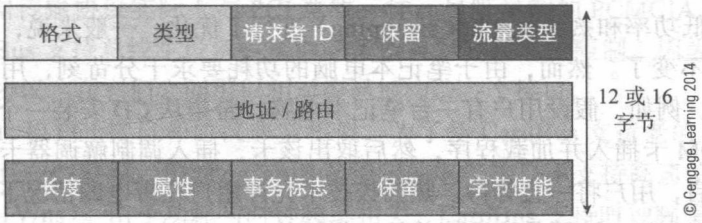


图 4-59 PCIe 报文头部的结构

4.7.3 CardBus、PC 卡和 ExpressCard

就像台式计算机的发展那样，便携计算机也在发展。20 世纪 90 年代笨重的可移动计算机导致了笔记本和上网本的出现。上网本本身受到了非传统计算机、平板电脑等挑战。这种进化需要相应的大容量存储器技术、性能和能耗有效的处理器技术、低功耗和高分辨率的显

示技术，以及接口技术的发展。本节将介绍接口是如何发展以允许小的外部设备与计算机系统总线相连接的。虽然这些接口可以用于台式电脑，它们主要用于笔记本。过去的上网本计算机一般来说不属于这个组。首先，它们太小，不足以支持所描述的接口类型。其次，在我看来，设计专用接口是一种退步（例如，Apple iPad 的接口）。最后，WiFi 的使用允许无线互连。接下来介绍基于 PCI 的计算机的接口，然后是基于 PCI express 计算机的接口。

引入 CardBus 和插入该总线的 PC 卡极大地增强了笔记本电脑的功能。简而言之，CardBus 是用于笔记本电脑的一个种扩展总线，它允许用户插入小的、信用卡大小的模块至笔记本电脑。这些模块提供了丰富的功能，从微小硬盘至无线 LAN 适配器，再到基于卫星的全球定位系统。当然，几乎与设计和制造外部设备的速度相同，它们从计算机的外部进入计算机的内部。例如，所有的笔记本电脑和上网本现在都包括内置的 WiFi，甚至在某些笔记本电脑中 3G 无线连接现在已经成为标配。计算机中将包含 GPS 以与 GPS 手机竞争。

PCMCIA（Personal Computer Memory Card International Association）是非营利性个人计算机存储卡国际协会的注册商标，该协会是一个贸易协会，以促进 PC 卡技术、创建技术标准、并提供信息[⊖]。一些文献将 PC 卡认为是 PCMCIA 卡，但从技术上讲这是不正确的，这是因为 PCMCIA 是贸易协会而不是卡。PC 卡提供的典型功能包括：

- A/D 转换和数据采集
- CD-ROM 接口
- 手机接口
- 以太局域网适配器
- GPS（全球定位系统）卡
- 红外无线局域网适配器
- 操纵杆接口
- 存储器（包括 Flash、SRAM 和硬盘驱动器）接口
- 调制解调器和以太网组合卡
- 调制解调器和 ISDN 卡
- 无线局域网适配器
- SCSI 适配器

CardBus 是 1996 年由 PCMCIA 制定的 32 位总线标准，主要用于便携计算机和笔记本电脑。它已经为低功率和热插拔（hot swapping）进行了优化。一般来说，用户配置好台式计算机后就保持不变了。然而，由于笔记本电脑的功耗要求十分苛刻，用户不希望负担不会被使用的功能。例如，假设用户有一台笔记本电脑，希望从 CD 安装一个程序，然后再上网。先将 CD-ROM 卡插入并加载程序，然后取出该卡，插入调制解调器卡。如果每次换卡都必须断电再重启，用户将一直等待。热插拔技术允许用户在带电的情况下拔掉卡并插入新的卡。从电的角度来说，这不是一项简单的工程任务。

虽然 CardBus 必须与其他现代总线接口（如 USB 总线和火线）相竞争，它确实有一个显著的优势。CardBus 的连接器既是接口又是设备扩展槽（device bay）。PC 卡插槽可以巧妙地隐藏复杂的插入功能，而不需要太多的外围设备接插线或者外部电源。一些制造商已经在 USB 上采用了这种方法，使其适应具有扩展的 USB 插头（例如安全设备和闪存系统）的一

⊖ PCMCIA 从 2009 年不复存在，因为 ExpressCard 从 2007 年开始已经取代了 PC 卡。

些系统。

PC 卡历史

与 PC 的所有其他方面几乎相同的是，CardBus 也发展进化了。一些早期的便携式计算机基于 ISA 总线实现 PCMCIA 卡，速率为 8.33MB/s。1985 年，日本电子产业发展协会（Japan Electric Industry Development Association，JEIDA）开始 PC 卡技术的标准化过程，到 1990 年发布了 4 个规范。个人计算机存储卡国际协会（PCMCIA）成立于 1989 年，由多家公司联合对存储卡进行标准化以促进贸易。

20 世纪 90 年代中期，发布了 PCMCIA 1.0 标准。它定义了卡的 68 个引脚的电气接口，以及 I 型和 II 型 PC 卡的形状因子。卡的形状因子及其 68 针的连接器最开始于 1985 年由 JEIDA 定义。原标准是设计用来为存储器提供接口的。到 1991 年 9 月底，PCMCIA 标准更新到 2.0 版本，它扩展了 68 针接口以提供 I/O 功能。标准 2.0 添加了对双电压存储卡的支持。到 1991 年年底，发布的 2.01 版本添加了 III 型卡和增强的软件接口。

接下来的升级是在 1993 年年中，其升级了 API 软件接口和卡信息结构（Card Information Structure，CIS）。1995 年，PCMCIA 卡消失了，出现了 PC 卡标准。该标准要求每个 PC 卡上都具有卡信息结构。1997 年，在便携计算机上实现了 32 位 33MHz 总线的 PCMCIA 5.0（它允许以 132MB/s 的速度进行 32 位数据传输）。

CardBus 卡配置软件被称为卡和套接字服务（Card and Socket Service），通过自动分配系统资源来提供即插即用功能。配置软件还实现了热插入，允许用户在不关闭电源和重启的情况下更换卡。套接字服务可以作为设备驱动程序的组成部分，也可以内置在 BIOS 中。卡服务软件层是一个位于套接字服务上层的应用编程接口。该 API 允许用户在 PC 卡内部集成 Internet 应用。

1. CardBus 卡

所有的 CardBus PC 卡都通过 68 针连接器连接主机系统。卡被金属保护层封装以提高其信号完整性。旧的 PC 卡有以 ISA 总线速度工作的 8 位或 16 位的接口。CardBus 提供了以频率高达 33MHz 和峰值带宽为 132MB/s 工作的 32 位多路复用地址和数据通路（64 位 PCI 总线提供的频率为 66MHz，峰值带宽为 528MB/s）。

CardBus 数据传输协议实际上与 PCI 总线一样。尽管早期的 PCMCIA 设备只能作为总线从设备，CardBus 支持总线主设备，这消除了所有对其功能的限制。因为 CardBus 和 PCI 总线共享相同的协议，在它们之间可以相对容易地构造一个接口。

虽然 CardBus 和 PCI 接口和协议非常相似，为了使 CardBus 支持老的 PCMCIA 卡，其接口和协议还是有一些差异。例如，CardBus 指定的电气接口不支持高速信号（即信号上升沿和下降沿大于 1V/ns）。由于功耗与时钟频率相关，CardBus 支持时钟频率的软件控制。当卡首先插入时检查卡的需求然后再调整电压，通过这种方法实现 5V 或 3.3V 的双电压操作。CardBus 标准已经为将来的版本中进一步降低工作电压进行了预先准备。

2. ExpressCard 卡

相对于约 2005 年出现的新一代高机动性、轻便的笔记本电脑来说，PC 卡是很大的。2003 年，出现了被称为 ExpressCards、以 2.5Gb/s 工作的一系列卡。2009 年出现了 ExpressCard 2.0，最大传输速率为 500MB/s。图 4-60 显示了两个 ExpressCard 以及 CardBus

卡。每个 ExpressCard 都有相同的连接器接口：一个版本为 34mm 宽，另一个为 54mm 宽。有趣的是，ExpressCard 支持两种接口：PCI express 总线接口和 USB 2.0 串行总线接口。ExpressCard 标准规定主机系统应该能够支持 3.3V、1000mA，辅助的 3.3V、250mA，以及 500mA、1.5V。使用 USB 总线（见后）时，该电流可以驱动 ExpressCard 模块。

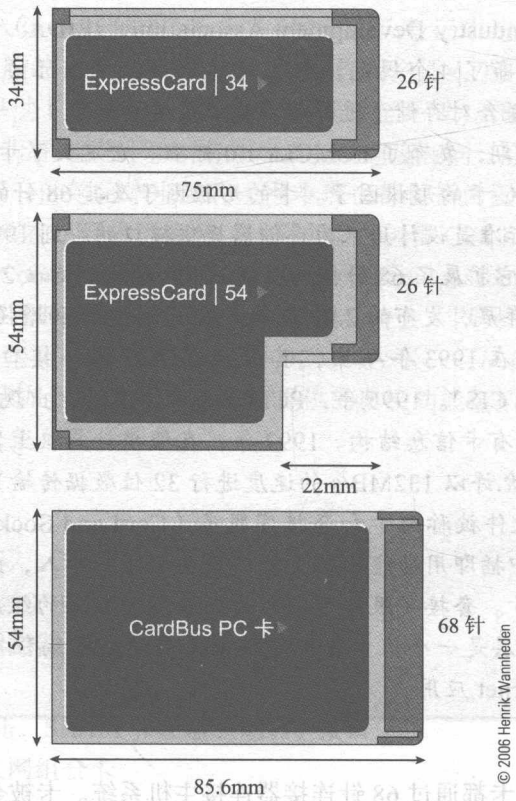


图 4-60 ExpressCard 外观

本章后面将讨论连接计算机和外部系统（例如打印机或者是其他计算机）的外部总线。首先介绍将高性能外围设备与计算机连接的 SCSI 总线。

4.8 SCSI 和 SAS 接口

最早设计用来连接计算机和外围设备的外部总线之一是 SCSI 总线，它已经被证明适应于这个变化的世界。曾经有一段时间，它是专业和高端系统中总线的首选。今天，面对非常低成本和高性能的总线，例如 USB 和火线，它的应用范围正在减小。本节讨论它是因为其在计算机发展中的作用和对其他总线设计的影响。本节还将介绍现代的串行连接的 SCSI（serial attached SCSI，SAS），它采用了低成本的串行技术。基于 SAS 的产品最早在 2005 年被使用。

小型计算机系统接口（Small Computer System Interface，SCSI）是一种可以追溯到 1979 年的 8 位并行总线，当时磁盘制造商 Shugart 试图为其硬盘系列产品寻找一种通用的接口。SCSI 总线是一种并行数据总线，它整合了为总线的预期用途而优化的信息交换协议，实现了磁盘驱动器和其他存储系统与主机计算机的连接。图 4-61 说明了 SCSI 总线的概念，它

最初被称为Shugart联合公司系统接口（Shugart Associates System Interface，SASI）总线。1981年，Shugart和NCR与ANSI合作规范SCSI总线，它在1986年成为X3.131-1986标准。

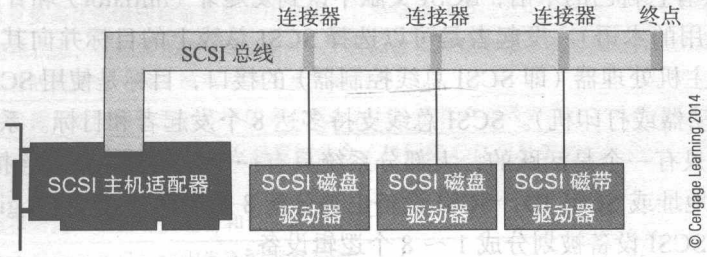


图 4-61 SCSI 总线

最初的SCSI-1总线工作在5MHz，允许最多7个外围设备连接在一起。人们提出了具有相同体系结构和不同级别性能的一系列SCSI总线（如表4-7所示）。1991年对规范进行了修订，提出了快速（fast）SCSI-2总线，它工作在10MHz，是具有16条数据路径这样宽（wide）的总线。Ultra SCSI或SCSI-3是下一步，它的时钟频率为20MHz。所有SCSI系统都支持异步数据传输，但SCSI-2还支持更快的同步数据传输。USB 3.0总线提供了4.8Gb/s或600MB/s理论极限。

表 4-7 SCSI 总线的版本

版本	宽度	数据速率 /MHz	吞吐率 /MB/s
SCSI-1	8	5	5
Fast SCSI	8	10	10
Fast Wide SCSI	16	10	20
Ultra SCSI	8	20	20
Wide Ultra SCSI	16	20	40
Ultra-2 SCSI	8	40	40
Wide Ultra-2 SCSI	16	40	80
Ultra-3 SCSI	16	80	160
Ultra 320 SCSI	16	160	320
Ultra 640	16	320	640

与8位和16位版本的SCSI总线一样，SCSI有3种电气接口。SCSI总线支持传统的单端（single-ended）电气接口，其中数据信号的电平参照0V的地线参考电平进行传输。长途传输将导致外部SCSI磁盘像天线那样动作，接收外来信号将导致错误和不正确的操作。比参考地线传输数据信号更好的机制是使用差分信号（differential signaling），其中数据以一对线（类似PCI express）间的电压来传输。差分信号需要两倍的连接数量，但更加不容易受到干扰，因为干扰在一对导线上增强了信号，但这并不影响相邻导线之间信号水平的差异。这种在两条导线上表现出来的抗干扰能力，称为共模抑制（common-mode rejection）。差分模式传输允许传输路径为25m，相比之下传统的单端SCSI系统的最大传输路径为3m。因为SCSI是总线，它需要在总线两端具有终止装置以防止反射（见本章的前面部分）。

另一种电气接口是低压差分（low-voltage differential，LVD）SCSI，它也使用了差分信号，但电压更低。低压信号技术提高了接收者的性能、降低了能耗，并允许接口与SCSI芯片集成。一些SCSI系统使用光纤通道（fiber channel），即使用光纤的串行接口。虽然光学系统需要数据转换器和并串（并行到串行）转换器，但它们对干扰高度免疫。

给出了SCSI的起源，那么SCSI总线已在中速至高速专业系统中成为受欢迎的磁盘接口就不足为奇了。SCSI磁盘驱动器包括内部SCSI接口，只需要一个连接器和将其连接至

SCSI 总线的导线。这些当然对 PC 中曾经流行但现在已经过时了 IDE 接口也是一样的。尽管高性能快速 wide SCSI PCI 卡相当昂贵，但适用于 PC 的 SCSI 总线接口卡非常普遍且价格低廉。

SCSI 总线具有它自己的术语，SCSI 文献中谈到发起者（initiator）和目标（target）（它们也是 PCI 总线适用的术语）。发起者是可以选择 SCSI 总线上的目标并向其发送命令的一种设备。发起者是主机处理器（即 SCSI 总线控制器）的接口，目标是使用 SCSI 总线的外围设备（如，磁盘驱动器或打印机）。SCSI 总线支持多达 8 个发起者和目标。系统可能具有多个发起者，但同时只有一个是活跃的，大部分系统只有一个发起者。每台连接到 SCSI 总线上的设备具有一个地址或 SCSI ID。SCSI 设备也被分配 8 个逻辑单元号（logical unit number，LUN）允许单个 SCSI 设备被划分成 1 ～ 8 个逻辑设备。

SCSI 总线操作使用消息传递（message-passing）原理，发起者告诉目标它想要执行什么操作。实现这种模式的原因是由于 SCSI 作为处理器与磁盘间总线的角色。目标需要相对长的时间来响应来自发起者的命令（例如，在磁盘上寻找一个给定的磁道）。此外，发起者传输的命令可能相当长，由 10 个或者更多的字节组成。SCSI 总线的特性是，若当前目标正繁忙地执行所分配的任务，总线可以被释放（release），从而被另一个设备使用。

表 4-8　SCSI-1 总线信号

引脚	功能
DB(0) ～ DB(7)	8 位数据总线
DB(P)	数据校验位
$\overline{\text{I/O}}$	数据方向的信号
$\overline{\text{REQ}}$	目标请求传输周期时使用的请求信号
$\overline{\text{ACK}}$	发起者完成数据传输时使用的确认信号
$\overline{\text{C/D}}$	发出控制 / 数据消息来指示控制或数据传输
$\overline{\text{MSG}}$	指出被传输的信息是消息的一个信号
$\overline{\text{BSY}}$	指出总线不是空闲的一个忙信号
$\overline{\text{SEL}}$	在总线选择操作中发出的一个信号
$\overline{\text{ATN}}$	由发起者在希望访问目标时发出的注意信号
$\overline{\text{RST}}$	将总线清空、重置所有发起者的复位信号

1. SCSI 信号

SCSI-1 总线由 18 根数据线、数据流控制线以及如表 4-8 所示的控制线组成，总线操作类似一个状态机（state machine）——任何时刻总线处于几个固定数量的状态之一。图 4-62 提供了一个简化的 SCSI 总线的状态图，表 4-9 描述了这些状态。

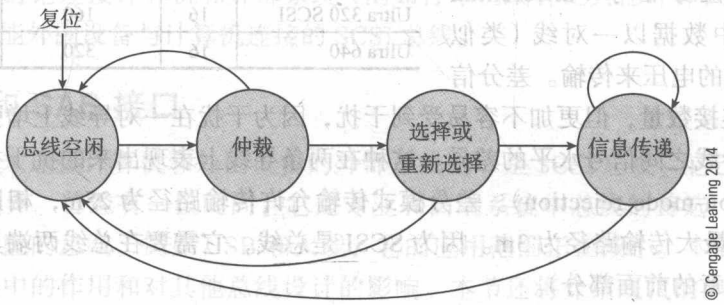


图 4-62　SCSI 总线的状态转换图

在总线空闲（bus free）状态，总线空闲，没有设备要使用它。这个阶段通过作废 $\overline{\text{SEL}}$ 和 $\overline{\text{BSY}}$ 信号来指示。在仲裁（arbitration）阶段，一个或多个设备试图控制总线。在选择或重新选择（selection or reselection）阶段，发起者使目标执行给定的任务。在信息传递（information transfer）阶段，数据在目标和发起者之间交换。

表 4-9 SCSI 总线状态

状态	描述
总线空闲	总线空闲状态，没有设备正在控制总线或者传输数据。该状态由作废 $\overline{SEL}$ 和 $\overline{BSY}$ 信号线来指示
仲裁	当设备希望控制总线并成为发起者，它通过发出 $\overline{BSY}$ 信号并将其 ID 放在数据总线上进入仲裁状态。如果没有更高优先级的设备需要使用总线，它将继续并通过发出 $\overline{SEL}$ 信号声明控制总线
选择	在选择状态，发起者选择（即指定）一个目标设备并发出命令使其执行特定的操作。通过将目标设备和发起者的逻辑或（OR）放到总线上来完成选择
重新选择	因为目标在长时间操作时能够放弃总线，在目标重新声明需要使用总线时需要重新选择状态
命令	目标设备使用命令阶段请求发起者的命令。目标将 $\overline{C}$/D 信号置为低来指示一条命令；将 $\overline{C}$/D 信号置为高来指示输出操作
数据	在数据阶段，数据在发起者和目标之间传输
消息	接口由发起者和目标之间发送的消息控制
状态	目标向发起者返回一个代码，表明操作的状态

2. SCSI 总线事务

下面介绍典型的 SCSI 总线事务。假设主机处理器希望从硬盘读取一个文件。处理器首先访问 SCSI 发起者，这当然是 CPU 和 SCSI 总线之间的接口。发起者几乎就是单个集成电路，对主机来说就是一个存储器映射的外围设备。

发起者检查 $\overline{BSY}$ 信号确定总线目前是否空闲。如果它是空闲的，发起者开始仲裁使用总线。SCSI 总线的 8 条数据信号线中的每一条都与 8 个仲裁级别（每个可能的 SCSI 总线用户对应一个）之一相关联。SCSI 总线支持 7 个外围设备——SCSI 总线发起者具有其自己的地址成为第八个设备。

请求设备所要做的就是在安装过程中向分配给它的数据线发出信号。被选中的数据线通过集电极开路输出为低电平有效，与 NuBus 仲裁的情况完全一样。如果参与仲裁过程的设备发现一个或多个高级别数据线被其他设备驱动为低，它将离开仲裁过程和并服从更高优先级的设备。

一旦发起者获得总线的控制权，它进入选择（selection）状态，允许其寻址或选择另一个总线上的设备。发起者发出选择信号 $\overline{SEL}$，以及要选择设备身份相对应的数据位和与自己身份相对应的数据位。例如，发起者的设备编号为 7，希望选择的设备编号为 3，发起者发出 $\overline{SEL}$ 信号并设置 DB(7) 和 DB(3) 为低。发起者现在可以释放 $\overline{BSY}$ 信号线。

被选中的设备（即目标）检测到它将被访问，并自己发出 $\overline{BSY}$ 信号来控制总线。发起者然后释放它使用的两条数据线，并作废 $\overline{SEL}$ 信号。注意，选择和重新选择（reselection）阶段是相似的。当目标原来被选择，目标的操作完成前被总线空闲阶段终止操作时，才会发生重新选择。一旦设备被选择或重新选择，总线进入信息传递阶段。该阶段其实就是以下几个操作之一：命令、消息输出、消息输入、状态、数据输出和数据输入。

通过使用 $\overline{REQ}$ 和 $\overline{ACK}$ 握手信号来控制数据流，现在可以在发起者和目标之间通过总线进行数据传输。$\overline{REQ}$ 请求数据传输，$\overline{ACK}$ 对其进行确认，就像前文描述的具有两根信号线的异步数据传输协议。这两个控制信号 $\overline{I}$/O 和 $\overline{C}$/D 确定数据流向和 DB(0) ~ DB(7) 上的信息是数据还是命令。起初，目标发出 $\overline{C}$/D 信号请求来自发起者的命令，发出 $\overline{REQ}$ 信号来触发数据传输（在这个阶段，目标知道其已经被寻址，但并不知道为什么）。发起者将一个字节放到数据总线上然后发出 $\overline{ACK}$ 信号，它是 $\overline{REQ}$ 的握手信号。然后根据对第一个命令字节的解释，目标可能会请求更多的命令字节。

当向目标发出长的命令时，发起者可以为其他总线业务释放总线，稍后再执行重新选择阶段。这使得SCSI总线在目标忙着执行其任务时变为空闲。

SCSI总线已经被证明是非常受欢迎的，因为它提供了高性能，并在USB、火线以及WiFi被广泛使用之前能够很好地适应中速、高速外围设备，如硬盘驱动器、光盘（CD）驱动器、打印机和光学扫描仪。

3. SCSI消息和命令

SCSI总线的发起者和目标之间的消息长度差异为1个字节、3个字节或者更多的字节。一个典型的由目标发送给发起者的消息是命令完成（command complete），表明命令已经完成，且一个状态字节已经返回给发起者。同样，来自目标的断路（disconnect）消息告诉发起者，连接将被挂起，稍后再连接。

SCSI总线也支持作用于设备本身（如磁盘驱动器和打印机）的命令。命令从发起者以顺序字节帧的形式传递。典型的SCSI命令格式包含以下信息。

操作码——定义操作的命令，一个字节。操作码还包括3位组代码用来表示命令的类型。

逻辑单元编号（LUN）——用来标识附加到目标上的设备，一个字节。该特性允许多个功能单元连接到一个目标。

逻辑块地址——可选值，2个或4个字节，为数据传输提供起始地址。

传输长度——可选参数，一个字节，定义了在读或写操作中被传输的逻辑块的数量。

参数表长度——用来配置操作LUN的参数表的长度。

分配长度——可选参数，用来指明发起者为返回的数据分配的最大字节数。

控制——一个字节，用来决定SCSI总线将如何表现。例如，它可以用来表明总线在当前命令后不应被释放，因为发起者将会发出另一个命令。

SCSI总线规范中实际的命令与磁盘驱动器、打印机、CD-ROM和其他外围设备的操作有关。然而，它提供了作用于所有外围设备的一些通用命令。例如，查询（inquiry）命令要求目标提供自己的以及相关外围设备的参数。

SCSI总线非常有趣，因为它模糊了总线与其接口以及连接至总线上的设备之间的区别。此外，它还包括一个专门为希望使用该总线的设备而制定的命令集。

串行连接的SCSI（SAS）

SCSI是一个沉重的负担——珍稀和值得尊敬但不是特别有用，直到20世纪90年代出现了USB和火线。然而，通过20年的发展已经扩展了SCSI和基于SCSI的系统，有些人并不愿意抛弃经过考验的技术，仅仅是因为它缓慢、笨重且昂贵。

串行连接的SCSI（SAS）试图保留SCSI的最好部分同时向USB和PCIexpress的世界靠近。串行连接的SCSI扔掉SCSI陈旧的物理层，将其替换为低成本、高性能的串行接口。这种串行接口提供比硬盘最大读/写速率快得多的12Gb/s的吞吐量。在2004年推出时，SAS支持3Gb/s，至2007年增加到6Gb/s，在2010年为12Gb/s。SAS是点对点的拓扑结构，与SCSI的多点总线不同。

SAS的物理层使用差动信号，支持长度为10m（33英尺）的导线。并行SCSI导线所需庞大的终止网络不再是必需的。SAS定义了两个低级层——物理层和PHY层，把传统物理层的功能（加上一些传统的链路层功能）分为两层。物理层只关心连接器和电

压水平，PHY 层关心数据编码、链接初始化、速度协商以及复位顺序。SAS 使用 8b/10b 编码。

SAS 的一个重要方面是，导线和连接器物理上与 SATA 接口兼容，SATA 接口现在已经被所有现代硬盘驱动器使用。因此，在传统的 PC 和基于 SAS 的系统上可以使用相同的低成本连接器。此外，SAS 支持 SATA 隧道协议（STP），使传统的硬盘驱动器可以连接到 SAS 体系结构中。

每个 SAS 设备在制造时都被分配了一个唯一的地址；也就是说，每个设备在被购买时都有其自己的地址。地址由名字地址权威（NAA，Name Address Authority）国际组织分配，它也负责海外 SAS 地址分配。

SAS 是技术融合（串行传输和 SCSI 网络）的一个很好的例子，也是计算机工业如何克服瓶颈（如，并行 SCSI 固有的速度缓慢问题）的一个很好的例子。

4.9 串行接口总线

曾几何时，串行总线用速度来换取简单性。并行总线需要多个数据路径和相应复杂的接插件安排及导线，而串行总线使用两条信号线来每次传输一位数据（bit at a time）：一条信号线传输数据，一条用于电回路。使用光纤电缆的串行数据链路需要提供单个数据路径。20 世纪 70 年代的 RS232C 串行连接速度缓慢，用户如果需要提高速度就必须使用并行数据总线。早期的 PC 具有使用 DB-25 连接器、由 8 条数据线组成的并行端口，用户可以使用 25 路带状电缆将其与打印机连接。现在某些打印机仍然还具有这样的一些基本并行接口，尽管它们越来越显得过时了。现代串行数据链接简单、快速、并且有效。

串行数据总线最大的优势是它的易用性。下面将介绍的总线[⊖]对计算产生了巨大的影响，因为它们为计算机与外部设备甚至是其他计算机之间的连接提供了低成本、高性能的解决方案。下面首先开始介绍对计算机通信产生深远影响的串行总线：以太网。

RS232C——噩梦

可能今天大多数计算机用户都不知道 RS232C 是什么，或者说曾经是什么。1969 年，电子工业协会（Electronic Industries Association）创建了两个不同设备间的接口标准：数据终端设备（data terminal equipment，DTE）和数据通信设备（data communications equipment，DCE）。这两个标准适用于调制解调器，它通过电话线以约 9600bit/s 的速度在计算机间传输数据。RS232C 早在个人计算机出现之前很长一段时间就已经开发出来了。

RS232C 接口使用异步串行传输（数据没有使用同步时钟，每个字符对应传输 8～11 个脉冲用来代表 ASCII 编码字符），一次只传输一位实现串行传输数据。数据连接器为 DB-25，具有 25 针的插头和插座（使用 25 根导线在两个数据通道传输数据，每个方向上的速率为 9600bit/s）。除了两个信号路径外，还包括几个控制信号，例如振铃指示器（ring indicator）。

⊖ 实际上，在讨论并行背板总线时已经介绍了串行总线 PCI Express，因为它在物理串行数据路径上建立了虚拟并行总线。

第一代微处理器系统使用现有的RS232C标准将计算机与其他设备（例如打印机）连接在一起。这种情况成为一个噩梦，因为RS232C的设计并没有预先的规划，导致的结果是接口通常需要手动设置计算机中的开关，甚至不得不在引脚间焊接导线。那时，即插即用（plug and play）的概念还没出现，用户不得不为每个连接到串行链路上的设备获取驱动。通常，简单地使打印机工作就能成为一项真正的成就。今天，用户只要把打印机和计算机通过USB电缆连接在一起，一切就搞定了。

4.9.1 以太网

本节通过简单描述以太网（Ethernet）来介绍串行总线，以太网是用来支持10Mb/s的局域网的。以太网可以追溯到1978年，现在已经成为IEEE 802.3标准。今天，它是低成本、工作在100Mb/s或1Gb/s的局域网标准。

在以太网中，所有设备都被连接到一根电缆上，不需要特殊的控制线。设备或节点，可以传输串行数据至连接了所有其他设备的公共总线。以太网传输电缆现在有4个可用版本：厚同轴电缆、细同轴电缆、低成本无屏蔽的双绞线以及光纤。表4-10定义了这些连接的命名法。10指的是链路的速度，Base指的是基带（baseband）传输（与调制（modulated）载波系统相对应），5/2/T/F指的是介质类型。术语1000BaseF指的事以太网介质使用光纤、工作在1Gb/s。

表4-10 以太网物理介质的命名约定

名称	媒体类型	最大分段长度/m	最大节点/段
10Base5	厚同轴电缆	500	100
10Base2	RG58（细同轴电缆）	185	30
10BaseT	UTP（双绞线）	100	1024
10BaseF	光纤	2000	1024

数据以包（packet）或帧（frame）的形式传输，如图4-63所示。一个以太数据包由7个字段组成，开始为8个字节（64位）的报头（preamble），用来使接收者的时钟与传输比特流同步。前7个字节的位模式为10101010。报头的最后一个字节为帧首定界符（start of frame delimiter），其特定模式10101011表明一个帧的开始。

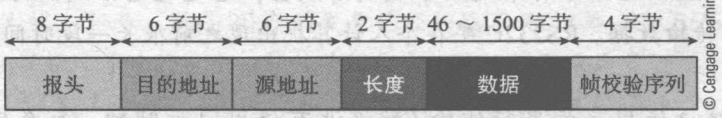


图4-63 以太帧

48位目的和源地址字段指出包从哪里来，到哪里去。长度字段定义了数据包的大小，长度为46～1500字节。然而，由于最小的数据字段必须包含至少46字节，比46字节少的数据字段需要被填充为46字节。最后，32位的帧校验序列提供了强大的错误检测码。

与并行总线不同，以太网只是一种消息交换机制（即没有控制信号）。以太网上的一个节点简单地将发给其他节点的帧放到以太网上。报文中的数据字段是如何解释的不属于以太网规范。以太网物理层使用的基带电缆，使用相位编码传输数据，速率为100Mb/s或1Gb/s。

基带这个词意味着直接传输数字数据而不需要调制解调器。

没有两个节点可以同时访问以太网并且它们的消息不会相互产生破坏性的干扰。当两条消息重叠时，发生了碰撞（collision），两条消息都会丢失。任意一个希望与另一个节点通信的节点只是简单地传输其消息。如果另一个节点也同时传输数据，或是在该消息完成传输之前发出，消息都会丢失。消息的丢失通过同样原始的技术发现：如果发送方没有在规定的超时时间内收到确认信号，它就认为消息在传输过程中被损坏。

当节点可以进行传输时没有任何的控制，因此没有什么可以阻止两个或两个以上的节点同时开始传输。最简单的竞争控制就是让发送者重新发送消息。但是这个方案行不通，因为竞争节点将重新发送消息，使得问题仍然得不到解决。

检测到碰撞的更好策略是回避（back off）或在重新传输帧之前随机等待一段时间。此后，竞争节点不太可能在完全相同的时间重新开始传输。以此种竞争控制工作的网络能够很好地适应突发流量。也就是说，只要平均流量很低（远低于总线的最大容量），这种安排就可以工作。如果流量上升，会出现这种情况：碰撞导致重复发出消息，会进一步产生碰撞并进一步出现重复发出消息，最终导致系统崩溃。

以太网的竞争控制机制允许节点在其试图发送帧之前侦听总线。如果有节点已经发出了消息，其他节点就不会试图发送。在以太网术语中，这就是所谓的顺从（deference）。只有当两个节点试图在几乎相同的时刻进行传输时才会发生碰撞。一旦节点开始传输，其信号传播至整个网络，其他节点不可以中断其传输。对几乎所有的系统，该危险地带（即消息从网络的一端传输到另一端的时间）非常小。

对该策略的进一步修改是让发送方在其传输时侦听总线。假设一个发送方开始传播，与此同时，另一个发送方也要开始传输。在很短的时间之后，两个发送方意识到总线正被使用并中止传输其消息。通过这种方式，发送方一旦检测到碰撞就停止传输，因此碰撞的影响被减少。一旦开始传输，它获得通道，经过网络端到端往返传播时间的延迟后，就可以确保一次没有碰撞的成功的传播。

以太网所采用的竞争机制被称为基于碰撞检测的载波侦听多路访问（carrier sense multiple access with collision detect，CSMA/CD）机制。当节点意识到其报文被另一个报文损坏，它通过传输拥塞（jam）报文来增强碰撞。如果它立即停止传输，其他发送方可能就不会检测到碰撞。碰撞可以由作为组成传输帧的一部分的错误检测码在很久以后间接检测到。这个过程效率低下且浪费时间。发送一个短的拥塞报文会使所有的侦听者发现碰撞。当发出拥塞报文后，在随机延迟后会试图进行下一次传输。如果多次尝试都失败了，随机延迟的时间会增加，发送方试图适应繁忙的通道。46 字节的最小数据包大小取决于检测碰撞的需要。

不像其他的通信系统，以太网依赖于消息的统计特性。有这种可能性，但不太可能，以太网节点由于总是被打断而从来没有发出一条消息。

以太网仍然通过调制解调器电缆连接电脑和局域网中的其他计算机。今天，打印机和存储器（称为网络附加存储器）可以连接到基于以太网的网络。近年来，WiFi 已经取代了家庭中基于以太网的网络，因为无线网络不需要有碍观瞻的电缆。然而以太网仍然继续发展，现在大多数 PC 都支持千兆以太网。10Gb/s 的以太网标准于 2002 年发布，它支持铜线连接和光纤连接。2010 年，IEEE 发布了 40Gb/s 和 100Gb/s 版本的以太网标准。

两种串行总线与个人计算机密切相关：USB 总线和火线（FireWire）总线。两种总线

执行类似的功能。USB 最初被设计为低成本、低速的总线，用来将诸如扫描仪和打印机之类的外围设备与计算机相连。而火线总线旨在支持更快的外围设备，如外部磁盘、摄像机。2000 年，更快的 USB 2.0 总线模糊了两者的区别。2010 年，出现了具有 USB 3.0 接口的系统，可以以 4800Mb/s 的速率工作。下面介绍火线总线。

4.9.2 FireWire 1394 串行总线

火线总线（现在称为 IEEE 1394 总线）是一个总线的实例，它开始作为公司的一个项目，后来成为一种行业标准。1986 年 Apple 公司开发了火线，作为并行 SCSI 总线的替代品，用于专业音频和视频。火线（FireWare）是 Apple 的商标——Sony 将相同的总线称为 iLink。

几个因素对火线总线的设计施以影响。第一个因素是成本（cost）。在过去的 40 年，微型计算机被嵌入到几乎所有的产品中，从电视机到洗衣机，在视听领域尤其如此。如果这些设备需要被连接，任意一种链接方式必须相对便宜——消费者不希望为一个插件支付一大笔钱。第二个因素是体积大小（size）。电子设备变得越来越小。当计算机比较大，其后部的连接器占用的空间并不重要。小型设备需要相应较小的连接器（例如，手持摄像机、电脑游戏控制端、MP3 播放器或者手机）。第三个因素是速度（speed）。每年，计算机的时钟频率和数字设备间的数据传输速度不断增加[⊖]。1975 年，600 波特率[⊜]的调制解调器被认为是快的；到了 1995 年，28.8Kb/s 的调制解调器通常用来作为计算机与 Internet 的接口；到了 2000 年，在许多家庭中可以发现 512Kb/s 的电缆调制解调器。今天许多计算机用户的电缆调制解调器以 20Mb/s 或更快的速度工作。第四个因素是可靠性（reliability）。系统通常由于连接器的故障而失效，这在很大程度上是因为重复插拔插头或不断扭曲电缆而造成的。可靠的连接器应该具有尽可能少的信号通路。

20 世纪 90 年代早期，最初称为 P1394 的高性能串行总线被提出（“P”表示临时标准）。相比于 SCSI 总线这样的并行总线，串行总线具有许多优点。特别是，串行总线只有两个连接器（如果使用光纤，只需要使用一个），这样降低了连线的成本以及连接器的成本和大小。P1394 总线利用现有的最佳技术来设计，可以支持多个不同的物理层（physical layers）；也就是说，P1394 总线并没有与某种类型的物理层实现绑定。串行总线的一些最重要的特征如下。

- 节点（即连接到总线的设备）地址的自动赋值，不需要地址开关或其他方式为节点分配地址。
- 变速数据传输，从 TTL 背板电缆的 24Mb/s 至有线介质的 400Mb/s。IEEE 1394b 规范将每个报文包含比特位的数量翻了一番，使得总线速度增加为 800Mb/s。
- 有线介质允许最多 16 个物理连接或有线跃程（cable hop），每个跃程最大为 4.5m。
- 公平的总线访问机制保证所有节点同等的访问权。

⊖ 当然，对该说法需要补充说明：处理器的最大时钟频率在约 2010 年后停止快速增长。然而，由于微体系结构的改进、专门的指令集和多核处理器的使用，使得处理器的性能持续提高。

⊜ 数据传输速率通常以每秒传输的位（比特）数来表示。波特率这个术语来自电报，并通过开关或信号的传输速率（即，每秒传输的码元数）来衡量。如果数据使用两级的二进制信号传输，数据传输速率和波特率是相同的。然而，如果使用八级信号进行传输，由于每个码元可以携带 3 位数据，数据传输速率将为波特率的 3 倍。

- 符合 IEEE Std 1212-1991、(ANSI) 微机总线的 IEEE 标准控制和状态寄存器 (Control and Status Register，CSR) 体系结构。
- 1394 串行总线限制总线上的节点数量为 63。然而，通过总线桥接多个总线的方法可以支持多达 2^{16} 个节点。

1394 串行总线规范从开放系统互连 (open systems interconnection，OSI) 借用 ISO 模型。特别是，串行总线标准使用了抽象层，每个抽象层完成特定的任务。1394 接口的三层接口本质上与 PCI express 总线是相同的。

物理层——该级定义了信息在总线上的传输方式。它关心的是介质的电气性能和信号传输技术。物理层链路采用其上层链路层的逻辑符号 (logical symbol)，通过接口传输这些符号，并将它们传输给另一个节点的链路层。

链路层——该级描述数据包在总线上的传输方式。链路层从物理层获取数据，完成数据帧处理、寻址和错误检测。链路层将数据传递给上层事务层。在串行总线术语中，完整的链路层操作包括：仲裁、包传输以及被称为子处理 (subaction) 的确认。

事务层——该级关心总线上互相通信的节点使用的端对端协议。一个节点的事务层从应用程序接收数据并将其传递到链路层。在另一个节点，对应的事务层从链路层获取数据并将其传递给应用程序。

图 4-64 描述了 1394 串行总线的分层协议。每层都提供了特定的服务。因为每层都与其上层和下层按照严格指定的方式通信，可以将任何意一层替换为执行相同功能的系统。也就是说，1394 串行总线是技术独立的 (technology independent)。例如，开发一种更快的串行路径，可以用新技术来实现物理层，而使用物理层提供服务的链接层和事务层不会受到影响。

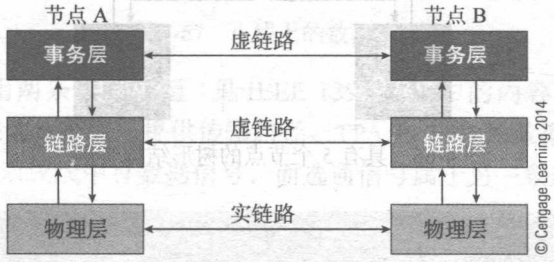


图 4-64 火线分层协议

1394 串行总线的总体结构如图 4-65 所示。1394 标准定义了两个串行总线环境。一个是在处理器系统内的背板环境 (backplane environment)，另一个是连接外部系统的电缆环境 (cable environment)。背板环境具有总线拓扑结构，工作速度为 25.576Mb/s 或 49.152Mb/s (取决于技术)。电缆环境支持 98 ～ 400Mb/s (1394a) 或者 800Mb/s (1394b) 的速率。这两个环境支持不同的物理层级拓扑结构、仲裁机制以及传输速率。2007 年发布的火线 S1600 和 S3200 版分别支持 1.6Gb/s 和 3.2Gb/s 的速率。

如图 4-65 所示，串行总线上的单个节点可能位于不同的背板环境中。从图 4-65 不太容易看出串行总线实际上是树形 (tree) 拓扑结构。图 4-66 显示了一个简单具有 5 个节点的树形结构。每个节点要么是分支 (branch) 直接连着多个邻居，要么者是叶子 (leaf) 只有一个邻居。许多串行总线应用菊链 (daisy-chain) 将节点以特殊的树形结构连接在一起。

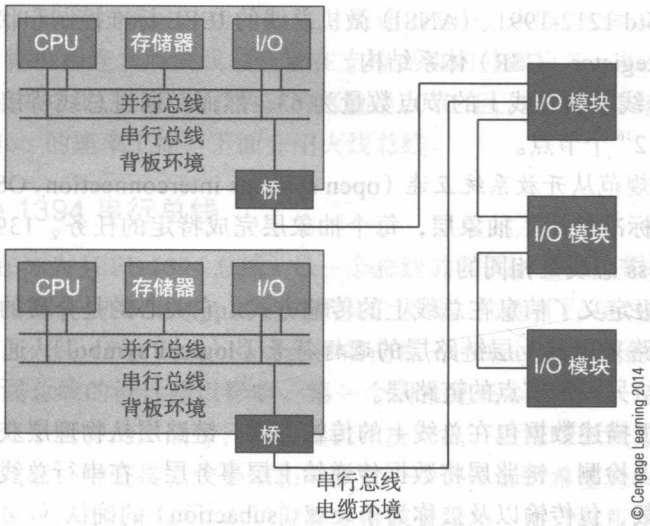


图 4-65 火线系统的结构

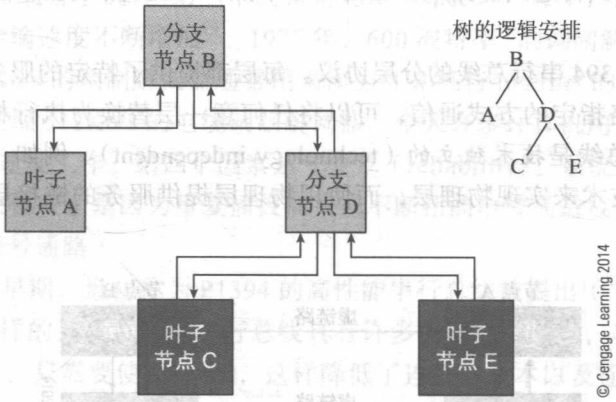
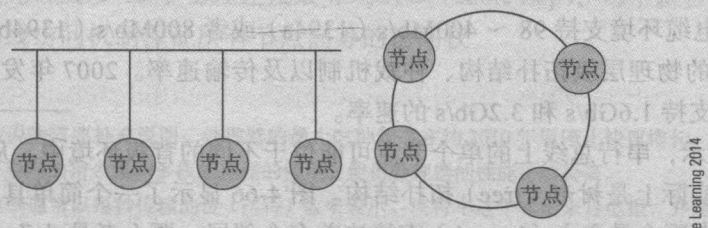


图 4-66 具有 5 个节点的树形结构实例

串行总线拓扑结构

数据传输系统由其拓扑结构决定其特点，拓扑结构描述了单个节点关联的方式。以太网是个总线，因为所有节点都通过它连接，信息通过总线从一个节点发送到其他所有节点，而不需要路由机制来确定信息如何在总线上传播。另一个系统是环形（ring）结构，其中所有节点彼此连接，信息从节点流动到节点。下图描述了总线和环形拓扑结构。



a）总线中所有节点都连接共同的数据通路

b）在环形结构中，每个节点仅与右手邻居和左手邻居连接

另一种拓扑是星形拓扑结构，它具有一个中央节点，所有流量都要经过该节点。所有其他节点都与该中心节点连接。

1. 串行总线寻址

支持多个节点的系统必须区分各个节点。1394 串行总线提供了 64 位寻址，每个地址的高 16 位代表节点的 ID（身份），支持 64K 个节点。串行总线将该 ID 划分为两个子字段：高 10 位指出总线 ID，低 6 位指定物理 ID 使每个电缆可支持 64 个设备。地址的其余 48 位分为寄存器空间、ROM ID 空间、初始单元空间以及初始存储器空间。这 48 位与具体实现有关；也就是说，它们可以根据需要来使用。

2. 物理层

这里不打算介绍物理层的细节。简而言之，串行总线以数据包的形式用半双工模式（即每次只能使用单向传输）传输信息。使用数据选通（STRB）信号来控制数据流。

数据使用不归零（non-return to zero，NRZ）格式传输，每当两个连续的 NRZ 值具有相同的值时 STRB 发生改变。这种机制可以很容易地使用数据和选通信号（通过异或运算）得到时钟信号。图 4-67 提供了数据信号序列 10110001 编码的例子。

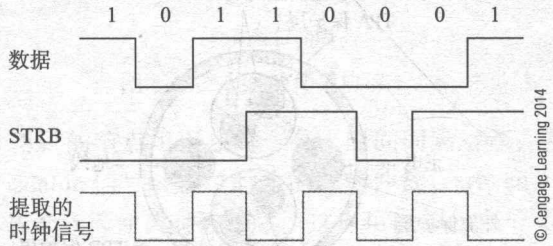


图 4-67 火线上的数据格式

1394 串行总线使用两条数据通道（见 IEEE 1394 附件中的内容）——TPA（双绞线 A）和 TPB（双绞线 B），来为每个方向提供传输路径。TPA 和 TPB 的数据路径和选通路径交叉。这样的安排不同寻常，双绞线中有数据信号，而选通信号属于另一条通道。

热插拔

当计算机启动并运行时，一些连接器不能删除或插入（例如，传统的 PS/2 键盘接口）。部分问题在于硬件。例如，插入一个连接器，插针不会在同一时间插入（对人来说，将卡插入插座好像是瞬时的，但插针插入时可能有 μs 级的差异）。这可能会导致不可靠的操作。同样，如果软件只在启动时轮询设备，删除一个设备或插入另一个设备将不可识别。

现代计算的无名英雄中的多数人认为，热交换（hot swapping）或热插拔（hot plugging）是理所应当的。以前，将新设备插入计算机或任何其他电子系统的唯一方法是，关机、插入、然后再开机。有时贴在设备上的贴纸用不友好的、大写的字母警告用户，当系统处于带电运行时试图插入东西都可能产生可怕的后果。如果用户把仍然冒烟的计算机送到维修店，他们会看着你，摇头说："你在系统带电时插入了什么东西，对不对？"

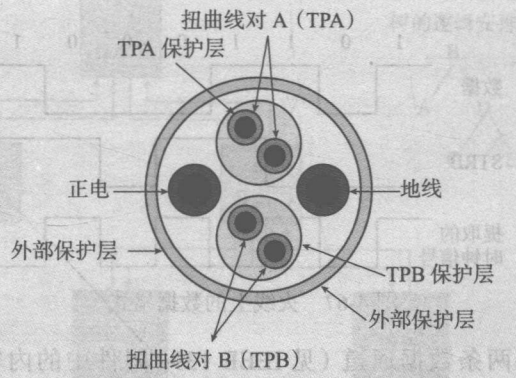
现在情况已经发生了变化。热插拔意味着用户可以带电插拔 USB 设备，而不必担心

产生什么后果。然而，由于必须处理电方面和逻辑方面的连接，这不是一件小事。必须设计连接器和插入处理机制使信号以正确的顺序连接。例如，接地连接通常是首先需要做的。某些引脚可能比其他的要长以确保首先被连接，比方说，地线。这样可以确保消除静电荷并向其余连接器提供参考电压。通常，信号以正确的顺序连接是重要的，且使用延迟电路来确保在剩余电路启动并运行后才完成关键连接。特别是三态总线驱动器在驱动总线时，在正确地激活它们之前必须关闭。

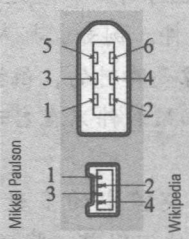
热插拔也可以应用到硬盘驱动器。这对高可靠的RAID存储系统来说非常重要，在不需要断电从而不会出现系统不可用的情况下，系统中失效的磁盘驱动器就可以被取出，然后添加一个新的驱动器。

IEEE 1394的硬件

1394总线使用六线（six-element）电缆。有两对扭曲在一起的数据导线（提供了两个独立的数据通道）以及两条供电线路。第7个导管环绕内部的6个导体以屏蔽整个电缆线。



有两种类型的1349总线连接器：四针和六针。六针连接器提供了完整的功能，而四针连接器缺少两个供电线。供电线可以提供8～40V、最高1.5A的电力（40V对于工作在3.3V或5V的大多数设备来说是较高的电压）。下图显示了这些连接器，比USB连接器的价格要昂贵。



IEEE 1394插头引线

信号	6针插头	4针插头	信号	6针插头	4针插头
火线	1	无	TPB+	4	2
地线	2	无	TPA−	5	3
TPB−	3	1	TPA+	6	4

3. 仲裁

串行总线实现了多种形式的仲裁（背板串行总线上的仲裁与电缆总线上的仲裁被区别对待）。这里介绍公平仲裁（fair arbitration），它发生在电缆总线上。该仲裁是基于地理位置（geographic）的仲裁，因为电缆上靠近根节点的节点总是获胜。

公平仲裁协议基于公平时间间隔（fairness interval）的概念，它由一个或多个总线活动时间间隔组成并由间隙分开。该间隙包括短的、被称为子处理间隙（subaction gap）的空闲时间和较长的、被称为仲裁复位间隙（arbitration reset gap）的空闲时间。在每个子处理间隙，总线仲裁决定下一个传输异步数据包的节点。图 4-68 说明了这个概念。

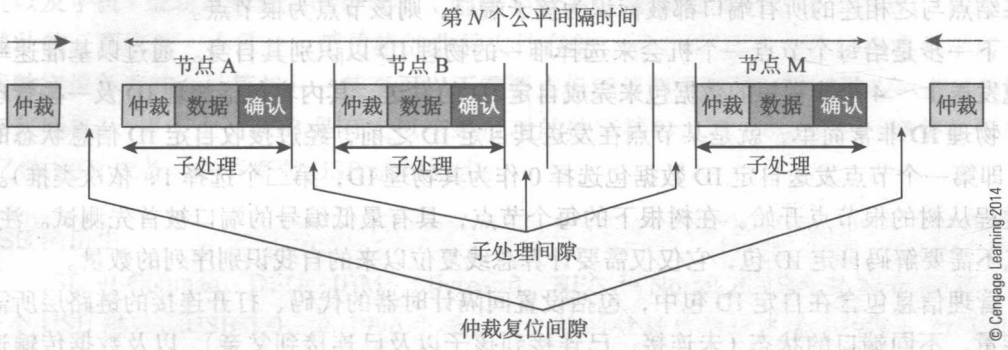


图 4-68　火线的仲裁协议

当使用公平仲裁时，活跃的节点可以在每个公平时间间隔启动，发送一次异步数据包。一个活跃节点只有当 arb_enable 信号被置位时才能进行仲裁。arb_enable 信号在仲裁复位间隙内被设置为 1，然后在节点赢得仲裁时被清零。这样可以禁止在公平时间间隔的剩余时间内进一步发出仲裁请求。当最终公平的节点完成仲裁后，公平时间间隔结束；此时由于所有节点现在都使其 arb_enable 信号复位并且不能驱动总线，因此需要 arb_reset_gap（仲裁复位间隙）。arb_reset_gap 重新使能所有卡上的仲裁，开始下一轮公平时间间隔。该过程如图 4-69 所示。

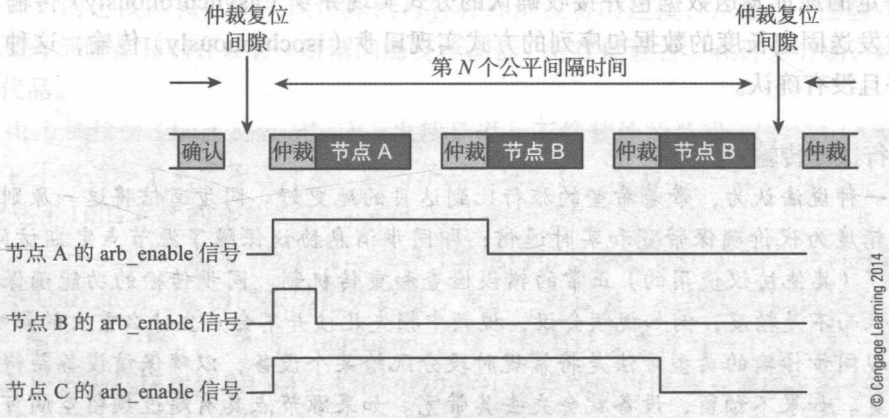


图 4-69　公平仲裁

背板环境还支持一种紧急仲裁的形式，它允许某个节点获得超过其公平份额的总线时间。然而，要求紧急仲裁的节点不允许长期使用总线。

4. 初始化

串行总线上的节点不需要开关来选择其地址，因为寻址可以被自动和动态处理。每当一个节点连接到总线，总线复位信号使得所有节点进入清除所有拓扑信息的状态。最初时，节点知道的唯一信息是它是一个分支节点、一个叶子节点、还是被隔离（非连接状态）的。

初始化后，串行总线拓扑被认为是一种树形结构，其中某个节点被当成是根节点，所有物理连接都有一个与其相关联的方向指向根节点。通过将每个连接的端口标记为*父亲端口*（parent，连接到离根节点更近的结点）或*孩子端口*（child，连接离根节点更远的结点）来设置方向。通过配置 tree-ID 进程中的超时（time-out）可以检测到拓扑结构中所有的循环。如果某结点与之相连的所有端口都被标识为孩子端口，则该节点为根节点。

下一步是给每个节点一个机会来选择唯一的物理 ID 以识别其自身。通过以基准速率向电缆发送 1 ～ 4 个非常短的数据包来完成自定 ID 的发送，其内容包括物理 ID 及一些管理信息。物理 ID 非常简单，就是某节点在发送其自定 ID 之前，经过接收自定 ID 信息状态的次数（即第一个节点发送自定 ID 数据包选择 0 作为其物理 ID，第二个选择 1，依次类推）。识别过程从树的根节点开始。在树根下的每个节点，具有最低编号的端口被首先测试。注意，节点不需要解码自定 ID 包，它仅仅需要计算总线复位以来的自我识别序列的数量。

管理信息包含在自定 ID 包中，包括设置间隔计时器的代码、打开连接的链路层所需要的能量、不同端口的状态（未连接、已连接到孩子以及已连接到父亲），以及数据传输速率的限制。

自定 ID 过程使用确定性选择过程，其中根节点将对介质的控制权传输给连接到它的且具有最小编号的连接端口，然后等待该节点发出 ident_done 信号，表明它和其所有的孩子都识别了自己。根结点然后将控制权传递给下一个具有最小编号的端口，然后等待该节点完成。当连接到根节点的所有端口的结点都完成后，根节点完成自我识别。子节点使用相同的方式以递归形式完成识别。

5. 链路层

链路层使用其下面的物理层的服务，在节点之间通过数据包的形式传输信息。数据可以通过向特定的地址发送数据包并接收确认的方式实现异步（asynchronously）传输。数据也可以通过发送固定长度的数据包序列的方式实现同步（isochronously）传输，这种方式简化了寻址并且没有确认。

同步串行数据传输

有一种说法认为，带着希望的旅行比到达目的地更好。同步通信将这一原则付诸实践，以精度为代价确保带宽和实时通信；即同步消息协议保障了源节点定期访问通道，但放弃了（其他协议使用的）正常的错误检查和重传机制。同步传输的功能确保了传输的连续性而不是精度，例如视频会议，视频中损失几帧并不会产生什么重大的影响。

实现同步传输的典型方法是将常规时段分配给某个设备，以确保该设备获得一部分可用带宽。如果不预留，设备就会失去其带宽。如果源节点具有超过预留空间可以存放的数据，则丢失数据。1394 串行总线和 USB 都支持同步通信协议。

异步和同步通信之间的根本区别是：异步协议（通过重传丢失的数据）保障传递过程和可靠性；而同步协议只保障时序，并不试图恢复丢失的数据。

串行总线使用的数据包包括具有源地址和目的地址的包头、数据包类型，以及错误检测码（即CRC）。

下一节将介绍通用串行总线（universal serial bus，USB）。最初，USB是火线的一种低成本的版本，以非常低的成本提供较低的数据传输率和缩减的性能。火线主要用于摄像机，USB主要用于键盘和鼠标。然而，USB标准的逐步改进给予了火线致命的打击。

4.9.3 USB

我认为，对个人数字革命（包括台式计算机、便携计算机、笔记本、MP3播放器、数码相机以及手机）成功贡献最大的是闪存（flash memory）和通用串行总线（USB）。闪存提供了健壮的、高密度、小尺寸、低价格的非易失性存储，而USB技术允许用户将几乎所有的现代数字设备连接至计算机——甚至可以不需要主机就将两个数字设备连接在一起（例如相机和打印机）。事实上，USB是有史以来最成功的数字接口，到2009年，已经售出了超过10亿件USB设备。本节将对USB进行概述。

USB的历史

USB由Compaq、DEC、IBM、Microsoft、NEC和Nortel在1994年联合开发。第一个USB规范，USB 1.0，在1996年提出，支持的数据传输速率为12Mb/s。USB 1.1在1998年发布，主要解决了集线器相关的问题。USB 1.1被广泛采纳。

2000年，出现了USB 2.0，它提供的最大数据传输率为480Mb/s。USB扩展至火线的传统领域，成为大多数PC与打印机、外部驱动器、键盘、鼠标等设备的接口事实上（defacto）的标准。

2009年，出现了USB 3.0，可以提供300MB/s（即2400Mb/s）的工作速度，取代了火线并不受其只能连接8个设备的限制。USB 3.0实现了对2.0版本的巨大飞跃，需要新的电缆格式和技术。

通用串行总线（USB）由许多公司联合研发，后来成为标准接口。从本质上讲，USB总线使用低成本的连接器和电缆实现计算机与各种外围设备的连接，从鼠标/键盘/打印机/扫描仪到如外部硬盘和闪存设备（所谓的随身驱动器）等存储设备。在许多方面，USB都是火线的替代品。

USB由主机控制（host controlled）；也就是说，不像其他的总线（如PCI、VMEbus或NuBus），它不支持多个主设备这种安排。每条总线只能有一个USB主设备（host，或主机）。图4-70描述了USB系统的分层星形（tiered star）拓扑结构。

图4-70中层次结构的顶层是主机，它与计算机通信并控制USB总线。主机连接到集线器，它是将USB总线分配给层次结构中更低层次的一种设备。集线器可以直接连接到一个外围设备（图4-70中标记为功能（function））、几个外围设备或者另一个集线器。每个集线器可能会连接到更低层次的集线器，如图4-70所示。集线器可能是一个单独的设备（例如，是一个USB端口扩展器，具有一个输入和多个输出），或者可以被内置到键盘、显示器、甚至是外部磁盘系统中。

1. 前两代USB

USB始于20世纪90年代中期，版本1.0和1.1提供的速度为12Mb/s（全速模式）和

1.5Mb/s（低速模式）。USB 2.0 版于 2000 年推出，提供 480Mb/s（高速），以及 12Mb/s 和 1.5Mb/s 这两种速度。USB 总线的 2.0 版本与早期版本兼容。由于 2.0 版本的性能远远胜于 1.1 版本，这里不讨论早期版本。

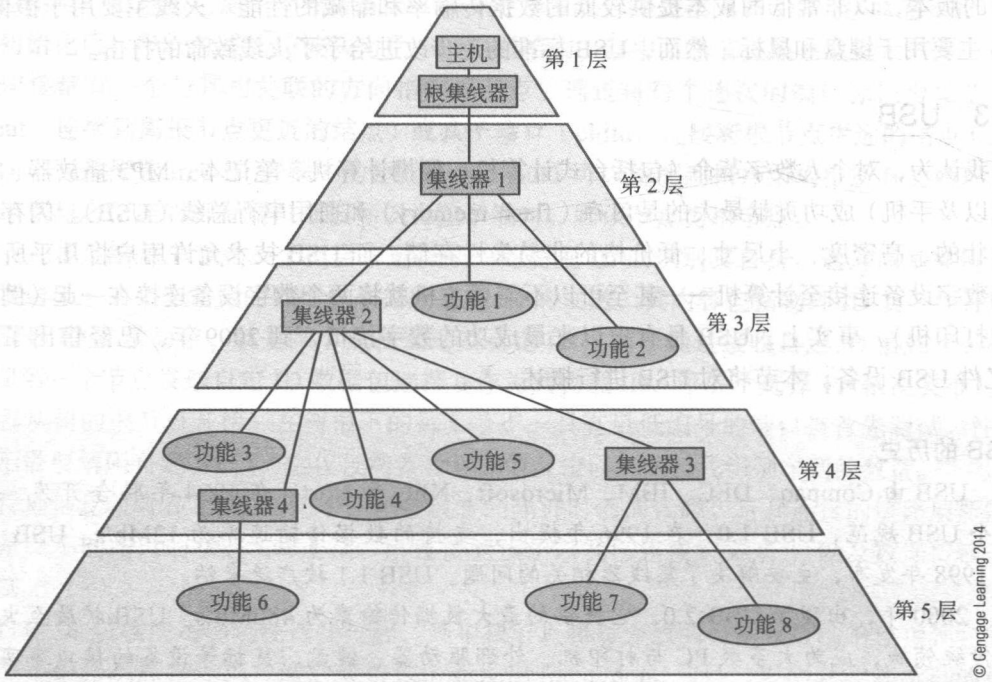


图 4-70 USB 总线拓扑

2. 电特性

USB 使用简单的多芯连接器与专用连接器相连。然而，由于 USB 的引入，外围设备变得越来越小，USB 电缆被迫遵循这一趋势，结果是现在有 4 种基本尺寸。图 4-71 显示了 USB 插头和插座。链路的计算机端使用 A 型插座和 B 型插头，使用这种实现方式的设备的数量相当可观。B 型插头和插座在链路的另一端使用（即在集线器或外围设备如打印机这一端）。迷你 B 型（Mini-B）插头和插座为数码相机、手机和便携式磁盘驱动器研发。图 4-72 给出了 USB 插头的照片，图 4-73 展示了 USB 系统的结构。

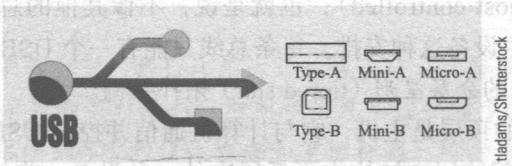


图 4-71 USB 连接器

USB 电缆使用 4 根导管。数据在标记为 D+ 和 D− 的一对双绞线中以不同的方式传输，如图 4-74 所示。前文曾谈到差动模式传输可以通过拒绝共模干扰来增加可靠性（在两条线上的感应电压不会影响导线之间的电位差）。双绞线被封闭在金属保护屏中，以进一步降低读取杂散信号[⊖]的风险。电缆的最大长度为 5m。当然，可以在 5m 处使用集线器来增加

⊖ 如果在 1.5Mb/s 的低速模式下使用 USB 连接，用户可以使用简单的最大长度为 3m 的非屏蔽线和非双绞线。

USB 路径的长度。

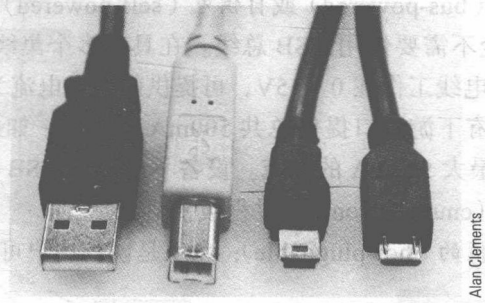


图 4-72 USB 连接器的照片

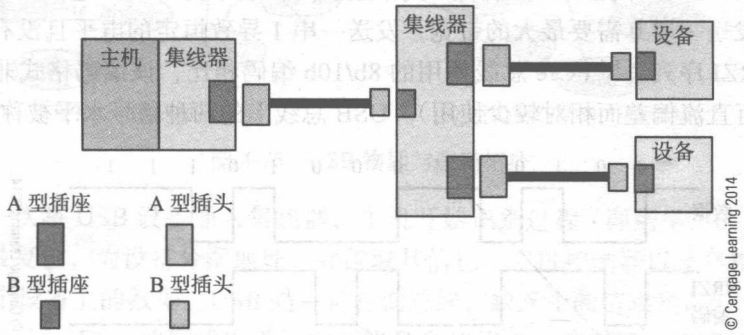


图 4-73 USB 机械环境

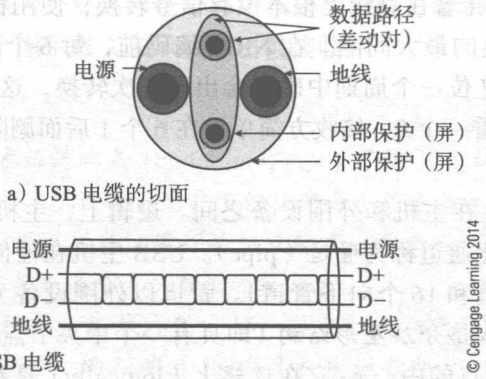


图 4-74 USB 电缆

更多的能量

USB 具有传输能量的能力，其目的是支持集线器和鼠标与键盘等外围设备。在实践中，许多厂家都利用了这一优势。如，充电器内置的 USB 可以为手机和 MP3 播放器充电。

一种被称为电池充电（battery charging）的新电源模式被添加至 USB 规范中。在这种模式下，主机可以在以 12Mb/s 通信时提供高达 1.5A 的电流，或者在以 480Mb/s 通信时提供 0.9A 的电流。此外，到 2010 年，世界上许多手机制造商都支持微型 USB 接口对其手机进行充电。

USB 电缆包括两个独立的供电线，为远程设备（例如，鼠标或键盘）提供电流[⊖]。USB 外围设备可以由总线供电（bus-powered）或自供电（self-powered）。今天，许多移动硬盘可以连接到 USB 插座而完全不需要使用 USB 总线。在具有多个集线器的 USB 系统中，必须管理能量的分配。USB 供电线工作在 0 ～ 5V，可提供的最大电流为 500mA。如果集线器是总线供电的，它可以为所有下游端口提供总共 500mA 的电流。如果集线器是自供电的，它可以为每个下游端口提供最大 500mA 的电流。设备不可以从 USB 获得超过 100mA 的电流，直到它已在一个称为列举（enumeration）的过程被主机识别。

USB 连接器是可热插拔的（hot-pluggable）；也就是说，用户可以在主机系统不断电的情况下插拔外围设备。

3. 物理层数据传输

在电平层面，USB 采用 NRZI 编码，其中逻辑 1 被表示为不改变电平，而逻辑 0 被表示为改变电平。发送一串 0 需要最大的带宽，发送一串 1 导致恒定的电平且没有信号转换。图 4-75 显示了 NRZI 序列（与 PCIe 总线使用的 8b/10b 编码相比，该编码格式非常古老，由于非自同步且具有直流偏差而相对较少使用）。USB 总线上的两种信号水平被称为 J 和 K。

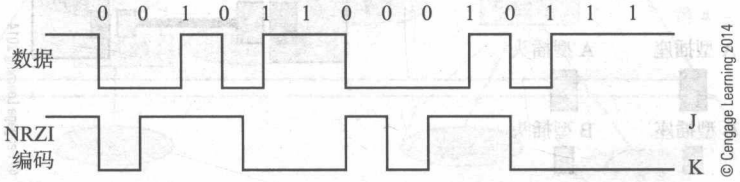


图 4-75 NRZI 编码

由于一长串的 1 意味着在总线上根本没有信号转换，使用称为位填充（bit stuffing）的技术可以确保电平转换的最大间隔。在 NRZI 编码前，每 6 个连续的 1 后将自动插入一个 0。在接收端，至少每 7 位一个周期中确保会出现一次转换，这足以保持收发同步。由于每 6 个连续的 1 后面都跟着一个 0，接收方简单地在 6 个 1 后面删除 0[⊜]。

4. 逻辑层

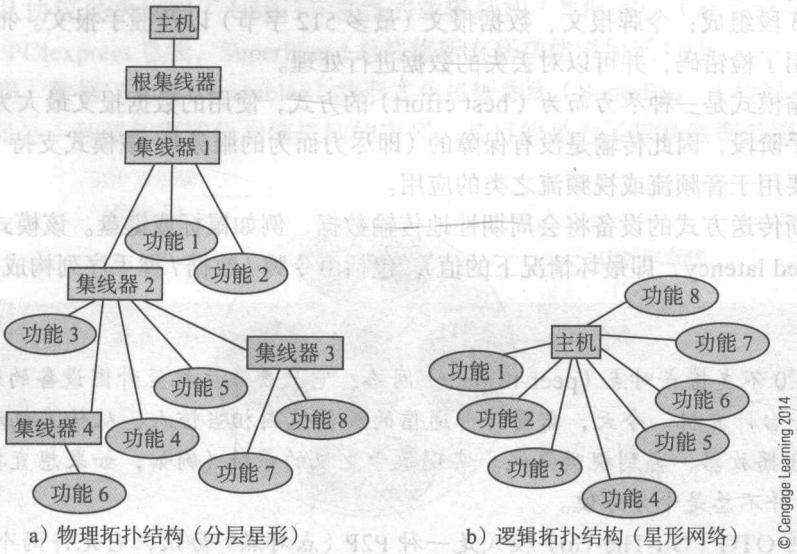
USB 总线通信发生在主机和外围设备之间。逻辑上，主机和设备之间存在通信信道，从 USB 的角度上说，此通道称为管道（pipe）。USB 主机在任何时刻都可以支持 32 个活跃的管道（16 个向上管道和 16 个向下管道）。管道以外围设备为端点（endpoint）终止。虽然 USB 系统的物理结构是分层星形结构（即具有一个中央节点，其他节点都与之相连，任意节点都可以是局部节点的中心），它在逻辑上（logically）是星形网络（star network），见图 4-76。所有节点必须通过单个中央节点相互通信。

更低的速度

USB 需要与多个设备间共享带宽，且为协议开销提供带宽。例如，虽然高速 USB 2.0 的理论最大数据传输速率为 57.3MB/s（480Mb/s），实际的速率为 20 ～ 25MB/s。

⊖ 火线有两个版本——支持供电；不支持供电。

⊜ 位填充最出名的是在 HDLC 传输中使用，它使用一串 6 个 1 作为令牌或标志。为了防止在数据流中出现令牌，每 5 个 1 个后面需要插入一个 0。在接收端，标志被清除（因为它们仅仅是 6 个 1 的序列），然后每 5 个 1 个后面的 0 被删除，由于它必定为一个填充的 0。



a）物理拓扑结构（分层星形）　　b）逻辑拓扑结构（星形网络）

图 4-76　USB 物理与逻辑拓扑

当用户第一次将 USB 设备插入集线器，主机开始识别过程（即列举，enumeration），该过程中新设备被复位，为设备分配地址，并读取其信息。主机控制器以主从模式工作，轮询 USB 网络中每个设备上的数据。USB 是一种轮询总线，缺乏中断请求机制。

信息以报文（packet）的形式发送，它可能是令牌报文、数据报文或握手报文。当主机发送（广播）一个令牌报文[⊖]时开始 USB 事务，该报文定义了当前事务的类型和方向、接收方的设备地址以及设备的端点号。然后根据令牌中指定的方向开始实际的数据传输。事务完成后，数据的接收方发送一个握手报文表示数据已被接收。

端点

与 USB 相关联的一个术语是端点（endpoint），它与 USB 的逻辑拓扑结构（星形）相关。每个 USB 设备逻辑上与主机连接，用户可以认为每个设备与主机之间都有一组管道（数据路径或通道）。这些数据路径通过设备的端点终止。需要说明的是，端点是保存来自主机数据的缓冲区，或准备发送给主机数据的缓冲区。

每个设备可能有 16 个端点，编号为 0 ～ 15。每个数据端点是单向的，要么发送数据给主机要么从主机接收数据。输入端点 IN 向主机发送数据，输出端点 OUT 接收来自主机的数据。控制端点为双向的，用于配置设备。每个设备必须指定端点 0 作为其控制端点。注意，实际上有 30 种单向数据路径，因为 1 ～ 15 个端点可以配置为 IN1、OUT1、IN2、OUT2…

USB 支持 4 种类型的数据传输：控制传输、批量数据传输、同步传输和中断数据传输。控制传输执行设备的配置和状态信息的读取等操作。批量传输模式是用于大批量数据的传

⊖ 令牌（token）这个术语来自具有环形拓扑结构的串行数据链接。为了避免冲突，只有一个节点具有称为令牌的报文。如果该节点需要继续工作，则必须具有令牌。如果它不想使用环，就将令牌沿着环路传递到下一个节点。该概念来自于单行线蒸汽机车的调度方法。单行线只允许有一个令牌（铃铛），工程师将用它来确保没有其他的机车可以使用这段轨道。

送，数据由3段组成：令牌报文、数据报文（最多512字节）以及握手报文。批量传输是可靠的，它采用了检错码，并可以对丢失的数据进行处理。

同步传输模式是一种尽力而为（best effort）的方式，使用的数据报文最大为1024字节。其中没有握手阶段，因此传输是没有保障的（即尽力而为的服务）。该模式支持24MB/s的最大带宽，主要用于音频流或视频流之类的应用。

使用中断传送方式的设备将会周期性地传输数据，例如鼠标或键盘。该模式具有有限的延迟（bounded latency，即最坏情况下的值），包括由令牌/数据/握手序列构成的3个阶段。

USB OTG

USB 2.0不支持点对点（peer-to-peer）网络；它只覆盖主机至外围设备的通信——主机通常是一台计算机。今天，需要外部通信的设备范围相当惊人，从传真机和复印机到手机、MP3播放器、再到视听系统。实现基于主机的系统（例如，如果想直接从手机上打印图像）并不总是十分方便。

USB的OTG（On-The-Go）模式是一种P2P（点对点）协议，它允许两个USB设备在没有主机参与的情况下通信，提供了设备间的直接通信。OTG操作于2001年12月被USB 2.0 a1修订版采纳。该USB的扩展版支持几个OTG设备之间或OTG设备和传统的USB主机之间的通信。该修订也引入了新的称为Micro-A和Micro-B的USB插头和插座。修订的标准引入了双重角色的设备（例如，OTG），它既可以作为主机也可以作为外围设备。此外，双重角色设备必须能够向总线提供至少8mA的电流（注意，大多数OTG设备是由电池供电的）。

当OTG设备作为主机时，可以仅仅支持目标外围设备列表（targeted peripheral list）；也就是说，OTG设备可能只与某些指定的外设一起工作，因此它并不是具有完整的、通用的USB功能的主机。OTG技术的重要组成是主机协商协议（Host Negotiation Protocal，HNP），它允许两个OTG设备之间传输控制。

5. USB 3.0

USB最彻底的改变发生在2010年——USB 3.0的引入，它使性能增加了10倍，且能耗更低。更重要的是，USB 3.0与USB 2.0兼容。USB 3.0十分有趣，因为它并不是USB 2.0的发展或延伸，而是与USB 2.0同时存在的替换总线（replacement bus）；也就是说，USB 3.0总线包含了USB 2.0总线。图4-77说明了USB 3.0电缆的结构。USB 2.0中的两个数据导管和两个供电线被保留。新增了两个新的差分导体对（SSRX和SSTX）来以全双工模式传输新的USB 3.0的数据。USB 3.0附加的功能被称为超速（SuperSpeed）总线，提供4.8Gb/s的最大速度。举例来说，USB 2.0需要13.9min来传输一部高清电影，而USB 3.0可以在70s内完成相同的传输。简而言之，USB 3.0是令人印象深刻的工程壮举，在功能和性能方面迈出了一大步，同时保持对现有巨大的USB 2.0用户市场的向后兼容。

USB 3.0与USB 2.0兼容，用户可以在主机上的USB 3.0插座中插入USB 2.0设备，该连接行为完全像一个传统的USB 2.0总线；也就是说，USB 3.0默认为USB 2.0。这种机制的关键在于插头和插座的安排。USB 2.0插头具有与插座匹配的小板，其一面有4个连接器。USB 3.0有两排连接器——一排与USB 2.0标准的完全匹配，另一排提供4个新的USB 3.0信号加上地线连接。换句话说，USB 3.0提供了真正的双总线机制，一个与USB 2.0相同，一个为新的SuperSpeed总线。

图 4-78 从协议层次说明了 USB 3.0 总线的逻辑结构（类似 ISO 开放系统互连的七层模型）。类似于 PCIexpress 总线，SuperSpeed 总线将所有新功能添加至 USB 3.0，它在物理层使用 8b/10b 编码。数据以扰码（scambled）的形式在超级总线（SuperBus）上传输。这不是一种安全机制，而是一种将数据转化为随机性的序列，其目的是为了提高数据链路的电气特性。

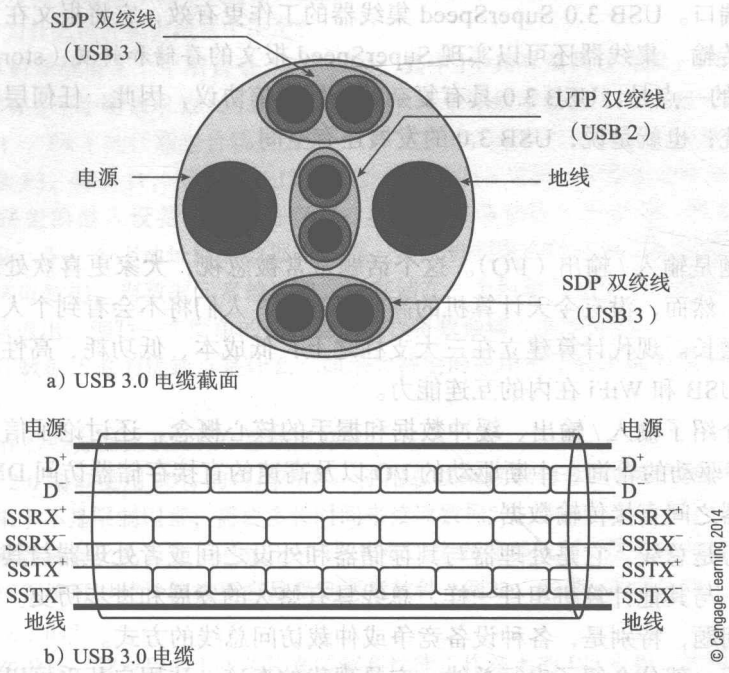


图 4-77 USB 3.0 电缆

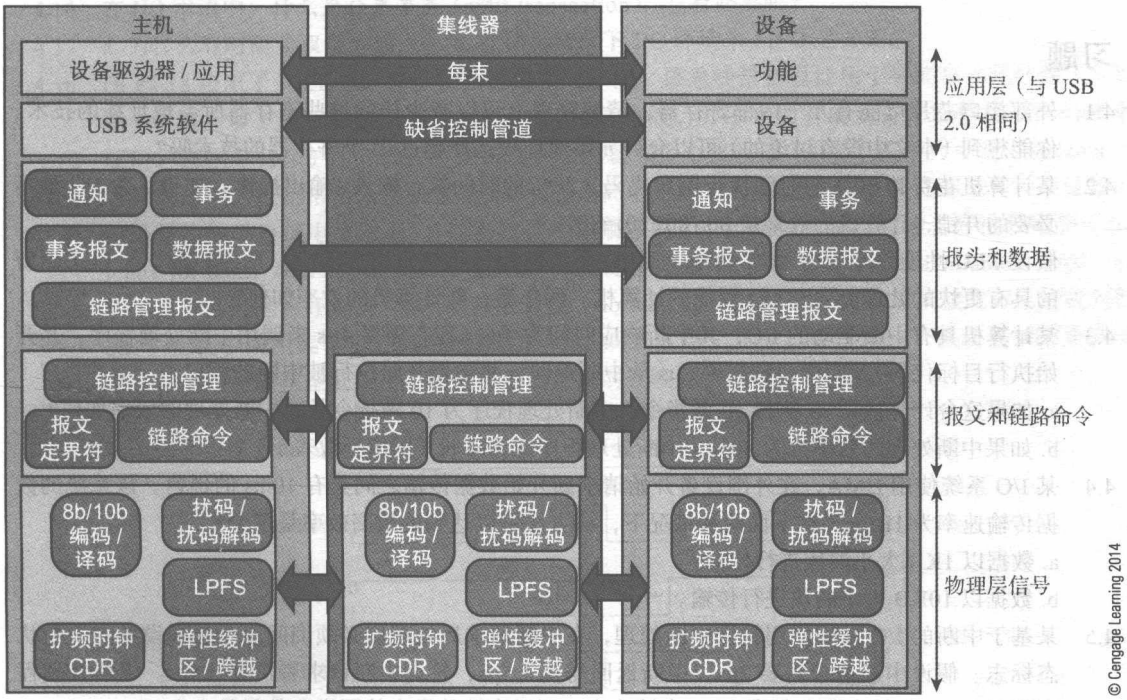


图 4-78 USB 3.0 协议层

图 4-78 演示了主机和设备之间通过集线器的端到端通信。USB 2.0 使用集线器扩展网络。用户可以购买具有一个连接至主机的输入和多个输出的 USB 集线器，或者可以使用这样的设备，例如，视频显示器可以具有连接至计算机的 USB 端口，另外还有 2 个或 3 个可以连接鼠标等设备的扩展 USB 端口。USB 2.0 集线器简单地将其输入端口上的所有消息传递到所有的输出端口。USB 3.0 SuperSpeed 集线器的工作更有效，它将报文在主机和参与通信的设备间进行传输。集线器还可以实现 SuperSpeed 报文的*存储和转发*（store and forward）。图 4-78 中重要的一点是，USB 3.0 具有复杂的分层通信协议，因此，任何层的改变或改进都可以不影响系统；也就是说，USB 3.0 的发展还有空间。

本章小结

本章的主题是输入 / 输出（I/O）。这个话题常常被忽视，大家更喜欢处理器和指令集体系结构的魅力。然而，没有今天计算机的高水平连接，人们将不会看到个人计算和无处不在的计算的快速增长。现代计算建立在三大支柱之上：低成本、低功耗、高性能的处理器，闪存，以及包括 USB 和 WiFi 在内的互连能力。

本章首先介绍了输入 / 输出、缓冲数据和握手的核心概念，还讨论了信息进出计算机的 3 种策略：程序驱动的轮询、中断驱动的 I/O 以及高速的直接存储器访问 DMA。DMA 在外围设备和存储器之间直接传输数据。

本章的核心是总线，它是处理器与其存储器和外设之间或者处理器与其他处理器之间的关键高速通路。与其他计算机组件一样，总线具有悠久的发展和进步历史。本章介绍了总线设计中的几个问题，特别是，各种设备竞争或仲裁访问总线的方式。

本章的最后一部分介绍了串行总线，它是现代的奇迹，让用户几乎可以将所有东西插入计算机而不必担心电气接口或数据传输方式。

习题

4.1 外部接口芯片可能有几个内部寄存器。请解释现有可以减少访问这些寄存器所需地址线的技术。你能想到（本文中没有讨论的）可以允许 n 条地址线选择超过 2^n 个寄存器的技术吗？

4.2 某计算机花费其 80% 的时间执行用户代码，20% 的时间执行输入 / 输出操作。如果认为 I/O 是不必要的开销，计算机的效率是 80/(80+20)=80%。

假设 CPU 性能每年综合增加 40%，而 I/O 系统性能每年综合增加 10%。两年以后，可以买到新的具有更快的处理能力和 I/O 性能的计算机。两年后，新计算机的效率如何？

4.3 某计算机具有中断驱动的 I/O，其中断响应时间为 4μs（即它需要 4μs 来调用中断处理程序，并开始执行目标代码）。同样，它需要 2μs 从中断返回，并重新开始执行被中断的程序。

a. 如果这台计算机每 μs 执行 10 条指令，中断处理程序为 10 条指令长，中断处理的效率如何？

b. 如果中断处理的效率约为 80%，中断处理程序应为多长（用指令数表示）？

4.4 某 I/O 系统使用 DMA，在外围设备开始请求到开始数据传输之间具有 10ms 的延迟。该系统的数据传输速率为 1B/μs。在下面两种情况下，该方法可以达到的最高效率是多少？

a. 数据以 1KB 大小的块进行传输；

b. 数据以 10KB 大小的块进行传输。

4.5 某基于中断的 I/O 系统使用轮询中断处理，在中断请求后处理器必须询问每个外围设备的中断状态标志。假设中断请求需要 2μs，中断返回需要 1.5μs，轮询中断请求者需要 0.5μs。中断处理程序需要 5μs。如果中断必须在 50μs 内被响应，可以支持的最大外围设备数量是多少？

4.6 该问题需要读者自己进行研究来了解 I/O 系统的性能变化。选取至少两种 I/O 总线，画出最大数据传输速率与时间的函数。

4.7 如果为地球上的所有人均匀分配以太网地址，每人都可以分到一个吗？如果人均有多个地址，每人将分配到几个地址？

4.8 多年前，某计算机被用于军事和航天应用。一个有趣的特性是，它没有任何形式的中断处理机制。你认为为什么设计者会做这个决定？

4.9 假设某计算机系统有 4 个中断设备，P1 ～ P4，其中 P1 具有最高优先级，P4 具有最低优先级。设计一个具有 4 个中断请求输入（P1 ～ P4）和 4 个中断请求输出（I1 ～ I4）的逻辑单元。该逻辑应该将 P1 ～ P4 上的任意组合转换为 I1 ～ I4 上反映出最高优先级输入的单个请求。例如，如果发出 P1 和 P3，输出 I1，不会输出 I2 和 I4。假设使用正逻辑（即请求信号为 1）。

4.10 某存储器映射的输入设备有两个寄存器：状态寄存器和输入寄存器。状态寄存器位于地址 PeriAddress，下一个字地址就是数据寄存器。输入机制查询状态寄存器的第 2 位，然后在获得状态位时读取数据。当数据元素被读取，它存储在一个简单的基于指针寻址的表格中。一旦输入 0 循环将退出。编写一个简单的循环查询过程将数据输入至表格中。

4.11 开环和闭环数据传输的优缺点是什么？如果为特定的应用程序必须选择某种方法时，你会考虑哪些因素？

4.12 USB 总线在哪些方面比 RS232C 总线要好？

4.13 高速 USB 2.0 端口使用 30% 的可用带宽。使用总线来读取包含 700 幅图像（每幅 25MB）的闪存卡。如果卡不是限制因素，需要多长时间来读取数据？

4.14 a. 某微处理器具有优先向量（prioritized vectored）中断处理机制。优先和向量化的含义是什么？
b. 中断处理系统可以使计算机更容易受到恶意软件（例如，病毒）的威胁吗？如果是这样，为什么？怎么发生的？

4.15 假设你正在设计一款专用的 I/O 芯片来缓解高性能工作站上的 I/O 负担。你将向计算机指令集添加什么特殊指令？如果有的话，需要增加哪些额外的硬件接口？

4.16 在 I/O 术语中，什么是分离事务（split-transaction），它有必要吗？

4.17 a. 为什么有时需要双缓冲输入？ b. 在什么情况下 I/O 环境中 FIFO 是必需的？

4.18 图 P4.18 给出了 IEEE 488 总线的数据传输时序图。该总线最初设计用于智能自动化仪器（实验室）环境。多个异步设备可以连接到 IEEE 488 总线。并不是所有的设备都能够以相同的速率运行。使用专利保护的三线握手来实现一个设备和多个接收器（488 术语中为听众“listener”）之间的通信。控制信号采用负逻辑（0 状态是高电平状态），由集电极开路门驱动（一个设备可将信号线驱动至低电平状态）。控制数据传输顺序的 3 个控制信号为没有准备好接收数据（not ready for data，NRFD，表明该设备当前尚无法接收数据）、数据有效（data available，DAV，表明发送者已经准备好数据）和不再接收数据（no data accepted，NDAC，表明接收方已经收到数据）。记住，使用集电极开路总线驱动实现了线或电路，其中如果任意一个驱动器被拉至低电平，信号线将被拉至低电平。请用自己的话说明三线握手是如何工作的。

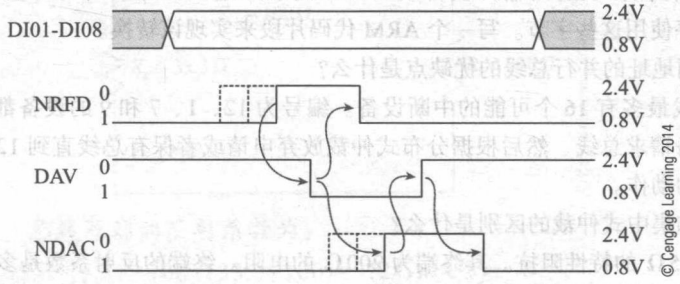


图 P4.18

4.19 图 P4.19 中的时钟电路是一个仲裁者。假设有两个设备，1 和 2，可以在任何时刻请求资源，如总线。问题是即使两个请求同时发生，也要确保只有一个请求者获得资源。设备 2 的电路有两个请求输入，加一个时钟和两个应答输出。说明电路是怎样可靠地执行该功能的。注意，该电路采用边沿触发触发器，不会产生亚稳定性（metastability）。

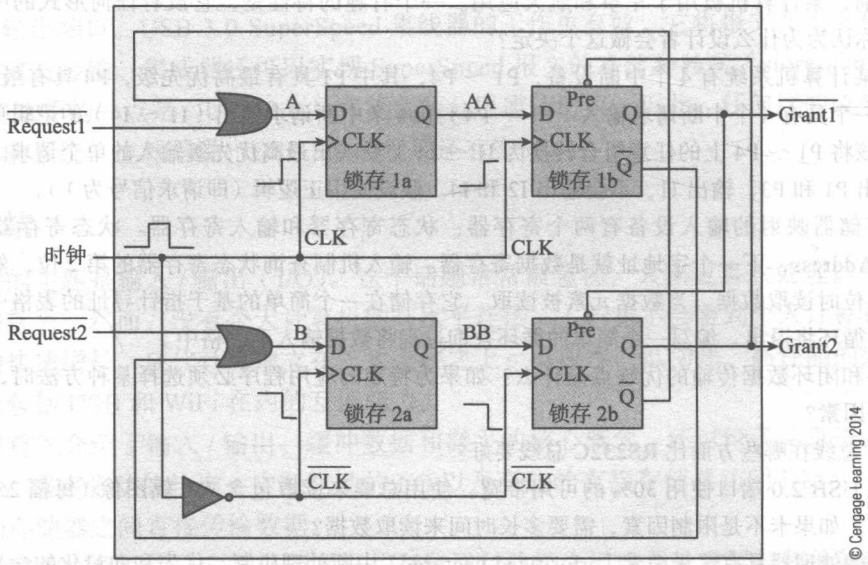


图 P4.19

4.20 某应用是 80% 计算密集型的，并占用计算机能力的 60%（包括 CPU 和 I/O）。假设由于更新，该应用程序每年将需要 20% 更多的资源；计算机处理能力每年增加 20%；而计算机 I/O 的性能每年提高 5%。5 年后，处理能力和 I/O 使用的情况将如何发展？

4.21 为什么 IBM 微通道结构在商业上失败了？

4.22 假定有一个系统，在两点间以 1024 字节的数据包（报文）传输数据。传输速率为 10Mb/s。在发送数据包之前，必须与接收者进行握手，这会带来 1ms 的延时。如果你决定重新设计系统，你可以减少 20% 的延迟或增加 10% 的速率，哪种选择更好？

4.23 某总线长 80cm，信号沿着它以真空中光速的 70% 传输。如果打开总线驱动需要 2ns，接收方需要 1ns 的建立时间和 2ns 的保持时间，执行一次端到端事务需要多长时间？

4.24 某山顶上的实验室位于基站 100 英里以外。与基站有两条链路：微波链路以光速传输数据，有线链路以光速的三分之二传输数据。为了安全，你决定在两条链路上传输普通数据流。由于传输速度的差异，基站在收到有线数据之前会收到微波数据。假设你决定在与有线数据比较之前对微波数据做一些错误处理。当每个操作需要 2ns 时可以执行多少操作？

4.25 ARM 环境中存储器映射的输入设备以 ABCD 的顺序返回 32 位 4 个字节的值。处理器需要以 DCBA 的顺序使用这些字节。写一个 ARM 代码片段来实现该转换。

4.26 a. 使用非复用地址的并行总线的优缺点是什么？

b. NuBus 总线最多有 16 个可能的中断设备。编号为 12、1、7 和 9 的设备都同时发出中断。说明每个设备请求总线、然后根据分布式仲裁放弃申请或者保有总线直到 12 号设备胜出时，总线上发生的动作。

4.27 分布式仲裁和集中式仲裁的区别是什么？

4.28 某总线具有 75Ω 的特性阻抗。其终端为 200Ω 的电阻。终端的反射系数是多少？

4.29 微处理器连接在传输线的一端。传输线有 100Ω 的特性阻抗。传输线由最近一端的微处理器驱

动，其阻抗为 50Ω。导线的远端有一个 300Ω 的电阻。

a. 总线每个端的反射系数是多少？

b. 在总线的远端，经过 t_{pd}（总线的端到端传播延迟）时间后的电平是多少？也就是说，计算当初始脉冲到达终点时总线远端的电压。

c. 如果微处理器发出 5V 的跨步电压，脉冲最初沿着总线传输时的信号幅度是多少？

4.30 解释 VMEbus 如何实现仲裁。

4.31 VMEbus 指定了属于 VMEbus 标准的特性，以及不属于该标准的特性（例如，总线释放算法的选择）。该策略的优点和缺点（即强制性要求和用户选项）是什么？

4.32 什么是原子的或不可分割的操作（或指令），该操作为什么有必要？

4.33 为什么机器状态在中断调用中需要保存，而在子程序调用中不需要保存？

4.34 解释为什么异步串行 RS232 接口可能是早期个人计算机遭遇的最大不幸。

4.35 某计算机的平均中断响应时间为 p，处理中断需要时间为 q，中断返回需要时间为 s。如果实现轮询中断机制，需要时间 x 来查询设备，时间 y 来服务中断的设备，在中断机制的效率低于中断驱动 I/O 之前，允许多少设备被查询？

4.36 为什么在实现存储器映射的 I/O 时，超标量处理器存在特定的问题。如何解决这个问题？

4.37 存储器映射的 I/O 可能无法在具有 Cache 存储器的系统中工作。为什么？补救措施是什么？

4.38 为什么要终止 SCSI 总线？注意，这同样适用于其他的总线。

4.39 总线的特性阻抗为 100Ω。必须用最高的电阻来终止总线。如果能容忍的最大反射脉冲为入射脉冲的 30%，可以使用的最大终端电阻是多少？

4.40 什么是嵌套中断？嵌套中断有什么优势和缺点？

4.41 以太网物理层协议代表开环还是闭环操作？

4.42 USB 2.0 数据链路接收到以下数据流。实际的源数据流是什么？
0001011111101111110111111011011111101111110111

4.43 PCI Express 总线使用的 8b/10b 数据编码的效率比使用其他数据编码的要低 20%。为什么还使用它？

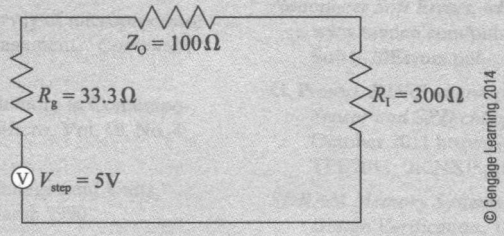
4.44 在计算机系统中 USB 和火线作为串行链路的优缺点是什么？

4.45 为什么不在一根电缆中结合 USB 和火线来实现双赢？你认为这是一个好主意吗？

4.46 本章并没有介绍蓝牙无线接口。调查该总线的性质。指出它的使用场合，以及相对 USB 这样的有线总线和 WiFi 这样的无线总线，它的优缺点。

总线反射的例子

本章前面只简单提到了总线的反射，本例用来展示总线反射的效果。下图中的总线，计算机传输线（即总线）从阶梯函数施加至近端开始（即 t=0），至四个单元延迟（一个延迟单元 t 为总线上端到端的传播时间）后，近端和远端的电压。



注意 Z_s=R_g。总线源端的反射系数为：

$(Z_s-Z_O)/(Z_s+Z_O)$=(33.3−100)/(33.3+100)=−(2/3)/(4/3)=−0.5。

总线远端的反射系数为：

$(Z_I-Z_O)/(Z_I+Z_O)=(300-100)/(300+100)=200/400=+0.5$。

沿着总线传输的初始脉冲由于源端电阻 R_S 和总线的电阻 R_O 而减小。如果产生的脉冲是 +5V，总线上的脉冲为 5×100/(33.3+100)=3.75V。现在可以考虑脉冲被反射后的行为。

3.75V 的入射脉冲到达远端并被反射。反射电压为 3.75×0.5=1.875V。因此，在总线远端的总电压是 3.75（入射电压）+1.875（反射电压）=5.625V。这大于用于驱动总线的脉冲。在电路开路的极端情况下，另一端的电压被加倍。

反射脉冲（即 1.875V）再次返回到源端并被反射。从源端被反射的部分为 1.875×−0.5=−0.9375V。远端的总电压现在为 3.75（原始信号）+1.875（输入信号）−0.9375（输出信号）=4.6875V。

每次连续的反射时，脉冲的振幅减少 50%（即源端 −0.5，目的端 +0.5）。信号穿过总线的时间为 t_{pd}。总线上连续传输的脉冲见下表。

时间	脉冲	近端电压	远端电压
$t=0$	+3.7500	3.7500	
$t=t_{pd}$	+1.8750		5.6250
$t=2t_{pd}$	−0.9375	4.6875	
$t=3t_{pd}$	−0.4688		4.2188
$t=4t_{pd}$	+0.2344	4.4531	

© Cengage Learning 2014

参考文献

第 1 章

Cache 存储器

J. R. Goodman, "Using cache memory to reduce processor traffic," *Proceedings of the 10th Annual International Symposium on Computer Architecture (ISCA)*, 1983

M. D. Hill, *Aspects of Cache Memory and Instruction Buffer Performance*, Technical Report CSD-87-381, UC Berkeley, 1987

M. D. Hill, "A Case for Direct-Mapped Caches, *Computer*, Vol. 21, No. 12, December 1988,

L. K. John and A. Subramanian, "Annex cache: a cache assist to implement selective caching," *Microprocessors and Microsystems*, Vol. 23, Issues 8–9, December 1999

L. K. John and A. Subramanian, "Design and performance evaluation of a cache assist to implement selective caching," *ICCD '97, IEEE International Conference on Computer Design*, 1997

N. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully Associative Cache and Prefetch Buffers," WRL Technical Note TN-14, Digital, Palo Alto, March 1990

F. Sebeck, *Instruction Cache Memory Issues in Real-time Systems*, Technology Licentiate Thesis, Department of Computer Science and Engineering, Mälardalen University, Västerås, Sweden, September 2002

A. J. Smith, "Cache Memories," *Computing Surveys*, Vol. 14, No. 3, September 1982

虚存

P. J. Denning, *Before Memory was Virtual*, http://cs.gmu.edu/cne/pjd/PUBS/bvm.pdf

P. J. Denning, "Virtual Memory," *ACM Computer Surveys*, Vol. 2, No. 3, September 1970

K. Elphinstone, S. Russell, and G. Heiser, *Issues in Implementing Virtual Memory*, University of New South Wales Report, UNSW-CSE-TR-9411, September 1994

B. Fuhrt and M. Milenkovic, "A survey of microprocessor architectures for memory management," *Computer*, Vol. 20, No. 3, March 1987

B. Jacob and T. Mudge, "Virtual Memory in Contemporary Microprocessors," *IEEE Micro*, Vol. 18, No. 4, July–August 1998

M. Milenkovic, "Microprocessor Management Units," *IEEE Micro*, Vol. 10, No. 2, March 1990

D. Roberts, J. Chang, P. Ranganathan, and T. N. Mudge, *Is Storage Hierarchy Dead? Co-located Compute-Storage NVRAM-based Architectures for Data-Centric Workloads*, HP Laboratories, HPL-2010-119, 2010

第 2 章

计算机存储器

Jon Burnett, *DDR3 Design Considerations for PCB Applications*, Application note AN111, Freescale Inc., July 2009

Calculating Memory System Power for DDR3, Micron Technical Note TN-41-01

Challenges and Solutions for Future Main Memory, Rambus White Paper, May 26, 2009 http://www.rambus.com/us/downloads/document_abstracts/products/future_main_memory_whitepaper.html

Design Guide for Two DDR3-1066 UDIMM Systems, Micron Technical Note TN41-08, Micron Technology Inc., 2009

A. Fazio and M. Bauer, "Intel StrataFlash Memory Development and Implementation," *Intel Technology Journal*, Q4, 1997

General DDR SDRAM functionality, Technical Note TN-46-05, Micron Technology Inc., 2001

B. Howard, R. Bacchus, E. L. Pope, and Br. Graham, *DDR3 For Dummies 2nd HP Special Edition*, Wiley, http://files.hypervisor.fr/doc/DDR3forDUMMIESv2.pdf

B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*, Morgan Kaufmann, 2007

K. Kilbuck, "Main Memory Technology Direction," Micron Technology, Inc., 2007

R. Mahajan, "Memory Design Considerations when Migrating to DDR3: Interfaces from DDR2," *DesignCon*, 2007

M. Muneeb, I. Akram, and A. Nazir, *Non-Volatile Random Access Memory Technologies (MRAM, FeRAM, PRAM)* http://www.imit.kth.se/info/SSD/KMF/2B1750/2B1750_06_RAMs.pdf

P. Murray and F. Al-Hawari, "Challenges in implementing DDR3 memory interface on PCB systems: a methodology for interfacing DDR3 SDRAM DIMM to an FPGA," *DesignCon 2008*, February 2008.

Nanometer Soft Errors, what lies beneath? www.tayden.com/publications/**Nanometer**%20**Soft**%20**Errors**.pdf

G. Prasad, *DDR3 migration to DDR4 - DIMM Thermal Sensor and SPD changes*, NXP Semiconductors, October 2011 http://sites.amd.com/us/Documents/TFE2011_012NXP.pdf

SDRAM Memory Systems:Architecture Overview and Design Verification, Tektronix, 2009

D. B. Stukov, G. S. Snider, D. R. Steward, and R. S. Williams "The missing memristor found," *Nature*, Vol. 453, May 2008

Two Technologies Compared: NOR vs. NAND, M-

Systems White Paper, July 03 91-SR-012004-8L

Understanding Soft and Firm Errors in Semiconductor Devices: Questions and Answers, Actel Corporation, Sunnyvale, CA, 2002

非易失存储器

J. Brewer and M. Gill, "Nonvolatile Memory Technologies with Emphasis on Flash: A Comprehensive Guide to Understanding and Using Flash Memory Devices," *IEEE Press Series on Microelectronic Systems*, 2008

B. Engel, "Technology, Manufacturing and Markets of. Magnetoresistive Random Access Memory (MRAM)," Everspin Technologies, Inc., Spintronics workshop, Kyoto, Japan, 2011 http://www.csis.tohoku.ac.jp/files/2011_SpintronicsWorkshoponVLSI_Japan_Engel.pdf

H. Li and Y. Chen, "An Overview of Non-Volatile Memory Technology and the Implication for Tools and Architectures," *Design, Automation & Test in Europe Conference & Exhibition*, 2009

NAND Flash 101: An introduction to NAND Flash and How to Design It In to Your Next Product, Micron Technical Note, TN-29-19, 2006

NAND vs. NOR Flash Memory Technology Overview, Toshiba, Inc.

Phase Change Memory (PCM): A new memory technology to enable new memory usage models, Numonyx White Paper

H. Pozidis, N. Papandreou, A. Sebastian, A. Pantazi, T. Mittelholzer, G. F. Close, and E. Eleftheriou, "Enabling Technologies for Multilevel Phase-Change Memory," *European Symposium on Phase Change and Ovonic Science*, Zurich, 2011

E. Ou and P. Leong, "Emerging Non-volatile Memory Technologies for Reconfigurable Architectures," *IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2011

S. R. Ovshinsky, "The Basis for Electronic Mechanisms in Ovonic Phase Change Memories," *European Symposium on Phase Change and Ovonic Science*, Zurich, 2011

T. Raja and S. Mourad, "Digital Logic Implementation in Memristor-Based Crossbars - A Tutorial," *DELTA '10 Proceedings of the 2010 Fifth IEEE International Symposium on Electronic Design, Test & Applications*, January 2010

The World is Flash, Denali White Paper, April 26, 2010, www.denali.com

P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," *International Symposium on Computer Architecture (ISCA)*, 2009.

第 3 章

辅存

80 mm (1,46 Gbytes per side) and 120 mm (4,70 Gbytes per side) DVD Re-recordable Disk (DVD-RW), Standard ECMA-338, December 2002 www.ecma.ch

J. Best, *The Femto Slider in Hitachi Hard Disk Drives*, Hitachi White Paper, 2007, www.hitachiGST.com

W. A. Burkhard and J. Menon, "Disk array storage system reliability," *The Twenty-Third International Symposium on Fault-Tolerant Computing*, FTCS-23, 1993

R. K. Chellappa and S. Shivendu, "Economics of Technology Standards: Implications for Offline Movie Piracy in a Global Context," *Proceedings of the 36th Hawaii International Conference on System Sciences*, 2003

P. M. Chen and E. K. Lee, "Striping in a RAID Level 5 Disk Array," *Proceedings of the 1995 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1995

R. Comerford, "Magnetic Storage: The medium that wouldn't die," *IEEE Spectrum*, December 2000

Deliver a superior digital archive solution for the media and entertainment industry, Business White Paper, Hewlett-Packard Development Company, 2011

S,.C. Esener, et al., WTEC Panel Report on The Future of Data Storage Technologies, International Technology Research Institute, World Technology (WTEC) Division, June, 1999

M. E. Fitzpatrick, *4K Sector Disk Drives: Transitioning to the Future with Advanced Format Technologies*, Toshiba America Information Systems, Inc., 2011 or at http://www.toshibastorage.com

Flash Memory Guide, Kingston Technology, http://media.kingston.com/pdfs/FlashMemGuide.pdf

R. Freitas, "Storage Class Memory: Technology, Systems and Applications," *Hot Chips Conference*, Stanford University, August 2010

Get SMART for reliability, Paper TP-67D, Seagate Technology, Scotts Valley, CA, July 1999

E. Grochowski and R. D. Halem, "Technological impact of magnetic hard disk drives on storage systems," *IBM Systems Journal*, Vol. 42, No. 2, 2003

Hard Disk Drive Specification Hitachi Ultrastar 7K3000, Hitachi Global Storage Technologies 2011

G. Herbst, *IBM's Drive Temperature Indicator Processor (Drive-TIP) Helps Ensure High Drive Reliability*, IBM Storage Systems Division, San Jose, CA, 1997

High Density Data Storage - Principle, Technology, and Materials, World Scientific Publishing Co. Pte. Ltd 1999 http://www.worldscibooks.com/materialsci/7014.html

IBM System x Server Disk Drive Technology, IBM Red Paper, 2011

In Search of the Long-Term Archiving Solution—Tape Delivers Significant TCO Advantage over Disk, Clipper Notes, report #TCG2010054LL, December 2010

K. A. S. Immink, *Codes for Mass Data Storage Systems*, 2nd Edition, Shannon Foundation Publishers, 2004

F. Marceteau, O. Rouchon, J. Raber, and G. Tsouloupas, *Media and Technology Appraisal for Long Term Preservation*, Partnership for Advanced Computing in Europe, 2011. www.prace-ri.eu

S. Mishra and P. Mohapatra, "Performance Study of RAID-5 Disk Arrays with Data and Parity Cache,"

Proceedings of the 25th International Conference on Parallel Processing, Vol. I, Architecture, 1996

R. New, *The Future of Magnetic Recording Technology*, Hitachi Global Storage Technologies, April 2008, www.asia.stanford.edu/events/spring08/slides402S/0410-Dasher.pdf

C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modeling", *Computer*, March 1994

Sony, metal particle and A3MP tape: Nanoscale technology for terabyte storage, Sony Electronics Inc., Park Ridge, NJ, January 2009.

Tang, Y. Lee, *Magnetic Memory: Fundamentals and Technology*, Cambridge University Press, 2010

R. Wood, Y, Hau, and M. Schultz, *Perpendicular Magnetic Recording Technology*, Hitachi White Paper, www.hitachiGST.com

Z. B. Zvonimir and H. V. Randall, "Storage Technologies," *Proceedings of the IEEE*, Vol. 96, No. 11, November 2008

光存储

80 mm (1,46 Gbytes per side) and 120 mm (4,70 Gbytes per side) DVD Re-recordable Disk (DVD-RW), Standard ECMA-338, December 2002, www.ecma.ch

Blu-ray Disc™ Format: 1.C Physical Format Specifications for BD-ROM, 6th Edition, White Paper, Blu-ray Disc Association, October 2010

DVD+ReWritable: How it works, Philips Disk Systems, 1999, http://www.dvdplusrw.org/resources/docs/howitworks.pdf

Gauch, S., "Impacts and dynamics of competing standards of recordable DVD media," *4th Conference on Standardization and Innovation in Information Technology*, September 2005

M. Hall and H. Takemoto, "Blue Laser Recording for High-Density Archival Storage," THIC meeting at the Raytheon ITS Auditorium, October 2004

K. A. S. Immink, "Shannon, Beethoven, and the Compact Disc," *IEEE Information Theory Society Newsletter*, December 2007

W. Kazuo, "Next-Generation Optical Disc Technologies," *Toshiba Review*, Vol. 66, No. 8, 2011

P. M. Lane and R.Van Dommelen, "Compact Disc Players in the Laboratory," *IEEE Transactions on Education*, Vol. 44, No. 1, February 2001

T. D. Milster, *Optical Data Storage*, Optical Sciences Center, The University of Arizona, Tucson, AZ, http://www.optics.arizona.edu/milster/380A%20Lab/Lab%203%20http://www.optics.arizona.edu/milster/380A%20Lab/Lab%203%20-%20CDROM/ODS%20Encyclopedia%20Article%20-%20Draft%20Version.pdf

T. D. Milster, "Status and Trends of Optical Data Storage Technology," THIC Meeting at the Center for Atmospheric Research, August 2007

B. H. Schechtman and S. Dror , "A Roadmap for Optical Data Storage Applications," *Optics and Photonics News*, Vol. 18, No. 32, 2007

K. A. Schouhamer Immink, "The CD Story," *Journal of the AES*, Vol. 46, 1998

W. Straw, "In Memoriam—The Music CD and Its Ends," *Design and Culture*, Vol. 1, No. 1, 2009

固态盘

N. Agrawal, V. Prabhakaran, and T. Wobber, "Design Tradeoffs for SSD Performance," *Proceedings of the USENIX Technical Conference*, June 2008.

L. M. Grupp, J. D. Davis, and S. Swanson, "The Bleak Future of NAND Flash Memory," *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, 2012

L. M. Grupp, A. M. Caulfield, J. Coburn, and S. Swanson, "Characterizing Flash Memory: Anomalies, Observations, and Applications," *MICRO'09*, New York, NY, December 2009

W. Hutsell, J. Bowen, and N. Ekker, *Flash Solid-State Disk Reliability*, Texas Memory Systems, November 2008, http://www.ramsan.com/files/f000252.pdf

J. Janukowicz and D. Reinsel, *Evaluating the SSD Total Cost of Ownership*, IDC, November 2007

J. Janukowicz and D. Reinsel, *MLC Sold State Drives: Accelerating the Adoption of SSDs*, MLC White Paper, September 2008

S. Kumar and R. Vijayaraghavan, *Solid State Drive (SSD) FAQ*, Dell, October 2011, http://www.dell.com/downloads/global/products/pvaul/en/solid-state-drive-FAQ.pdf

S. Lee, B. Moon, C. Park, J. Kim, and S. Kim "A case for flash memory ssd in enterprise database applications," *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 2008

B. Oliver, *Comparison of 32GB SSD vs 10,000 RPM HDs*, http://www.xlr8yourmac.com

OLTP performance comparison: solid state drives vs. hard disk drives, Principles Technologies Test Report, 2009, High Endurance Technology in the Intel® Solid-State Drive 710 Series, Intel Technology Brief, 2011

Solid State Drives Separating Myths from Facts, Toshiba America Electronic Components, Inc., December 2008, http://www.toshiba.com/taec/news/media_resources/docs/SSDmyths.pdf

The Top 20 Things to Know About SSD, Seagate Technology Paper, www.seagate.com/docs/pdf/ssd_faq.pdf

Wear Leveling Techniques in NAND Flash Devices, Micron Technical Note TN-29-42

RAID

P. M. Chenand D. A. Patterson, "Maximizing performance in a striped disk array," Proceedings of the 17th Annual International Symposium on Computer Architecture (ISCA), 1990

R. H. Katz, "RAID: A Personal Recollection of How Storage Became a System," *IEEE Annals of the History of Computing*, Vol. 32, N. 4, October–December 2010

D. A. Patterson, P. Chen, G. Gibson, and R. H. Katz, "Introduction to redundant arrays of inexpensive disks (RAID)," IEEE Computer Society COMP-

CON, February 1989

M. Staimer, *Ignore the Impending RAID Catastrophe At Your Own Risk*, White Paper, Dragon Slayer Consulting, 2011

L. Xiao, T. Yu-An, and S. Zhizhuo, "Semi-RAID: A Reliable Energy-Aware RAID Data Layout for Sequential Data Access," *27th Symposium on Mass Storage Systems and Technologies (MSST)*, Denver, CO, 2011

第 4 章

输入 – 输出（基础）

A. F. Harvey, *DMA Fundamentals on Various PC Platforms*, National Instruments, Application Note 011, April, 1991

FIFO Architecture, Functions and Applications, Texas Instruments Application Report SCAA042A, March 1999

R. Finger, *Using TI FIFOs to Interface High-Speed Data Converters with TI TMS320 DSPs*, Texas Instruments Application Report SDMA003, June 2001

Implementing FIFO Buffers in FLEX 10K Devices, Altera Application Note AN 66, Altera Corporation, January 1996

Implementing DMA on ARM SMP Systems, Application Note 228, Document number: ARM DAI 0228 August 2009

T. Jackson, *Advanced Bus-Matching/Byte-Swapping Features for Internetworking FIFO Applications*, Texas Instruments Application Report SCAA014A, March 1966

J. B. Johnson, "Application of an asynchronous FIFO in a DRAM data path," Thesis, College of Graduate Studies, University of Idaho, December 2002

K. Kittrell, *FIFO Solutions for Increasing Clock Rates and Data Widths First-In, First-Out Technology*, Texas Instruments, Report SZZA001A, 1996

B. T. Tan, "Generalized protocol for parallel-port handshaking," *Microprocessors and Microsystems*, Vol. 13, No. 9, November 1989

S. M. Taylor, "Data-driven handshake circuit synthesis," Thesis, School of Computer Science, University of Manchester, 2007

Using TI FIFOs to Interface High-speed Data Converters with TI TMS320 DSPs, Texas Instruments Application Report SDMA003, June 2001

总线

D. Abbott, *PCI Bus Demystified*, LHH Technology Publishing

J. Ajanovic, *PCI Express (PCIe*) 3.0 Accelerator Features*, Intel Corp., 2008, http://www.intel.com/content/dam/doc/white-paper/pci-express3-accelerator-white-paper.pdf

D. Anderson, J. Dykes, and E. Riedel, "More than an interface - SCSI vs. ATA," *Proceedings of the 2nd Annual Conference on File and Storage Technology (FAST)*, March 2003

J. Brewer and J. Sekel, *PCI Express Technology*, Dell White paper, February 2004

A.M. Caulfield, J. Coburn, T. I. Mollov, A. De, A. Akel, R. K. Gupta, A. Snavely, and S. Swanson, "Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing," *SC10*, New Orleans, November 2010

M. Davidsaver, *Understanding VME Bus*, https://pubweb.bnl.gov/~mdavidsaver/understanding-vme.pdf

R. Dominguez and T. Colligan, *SCSI vs. ATA: Interface Comparison*, Dell Technology Brief, December 1999

FireWire Design Guide, 1394 Trade Association, FWDG 1.0OTA 03/02/2010, 2010

FireWire™ Reference Tutorial (An Informal Guide), 1394 Trade Association, Mukilteo, WA, January 2010

A. Goldhammer and J. Ayer Jr., *Understanding Performance of PCI Express Systems*, Xilinx white paper WP350 (v1.1), September 2008

An Introduction to the Differential SCSI Interface, Texas Instruments Application Note 904, 1998

The GPIB (IEEE-488) Bus, www.optics.arizona.edu/opti680/uLab%20Week%2011.pdf

PCI Express - An Overview of the PCI Express Standard, National Instruments, 2009

PCI Local Bus Specification, Revision 2.3, PCI Special Interest Group, March 2002

NuBus Specification, Texas Instruments document TI-2242825-0001, Irvine, CA, 1983

C. E. Stevens, *USB Attached SCSI Protocol (UASP)*, USB implementer's forum, www.usb.org

B. G. Taylor, "Interfacing to NuBus," *Eurobus Conference*, Munich, May 1989

ULTRA2 SCSI, White Paper, LinFinity Microelectronics, Garden Grove, May 1996

Universal Serial Bus Specification, Compaq, Hewlett-Packard, Intel, Microsoft, NEC, and Philips, September 2000

Universal Serial Bus 3.0 Specification, Hewlett-Packard Company, Intel, Microsoft, NEC, ST-Ericsson, and Texas Instruments, June 2011

VMEbus Specification Manual, http://agata.pd.infn.it/LLP_Carrier/optoisolatore/Tundra/Doc/VME%20bus%20specifications.pdf

VME Technology FAQ, VITA, 2011, www.vita.com/home/Learn/vmefaq/vmefaq.html

M. Wolf, *Computers as Components: Principles of Embedded Computing System Design*, Morgan Kaufmann, 2012

Xilinx PCI Tutorial, Xilinx Inc., 2000

计算机存储与外设

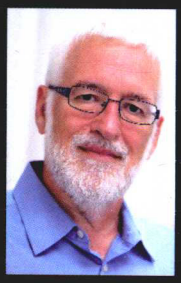
Computer Organization and Architecture Themes and Variations

本书由资深的计算机体系结构教育家Alan Clements博士编写，原书名为《计算机体系结构：原理与演变》（Computer Organization & Architecture: Themes and Variations），书中不仅覆盖单机系统的组成原理和系统结构的各个方面，还包括计算机的性能评价方法以及多发射、粗粒度并行等内容。作者希望本书能够适合电子工程（EE）、电子与计算机工程（ECE）、计算机科学（CS）等不同专业的教学需要。书中围绕基本概念、指令集体系结构、处理器组成和能效、存储与外设以及处理器级并行等五个核心问题将这些内容有条不紊地组织在一起，以便满足不同专业的教学需要。

中文版引进的时候综合考虑国内高校“计算机组成与结构”或类似课程的教学目标以及我们对本书的定位，对原书进行了适当裁剪和重新组合，分为两册：《计算机组成原理》和《计算机存储与外设》。

本书即为《计算机存储与外设》，涵盖原书第四部分，共4章，主要讲述计算机系统中的存储器、总线和输入/输出等内容。

作者简介



艾伦·克莱门茨（Alan Clements）国际著名的计算机体系结构教育的推动者和践行者。他于1997年获得英国拉夫堡大学（Loughborough University）博士学位，随后加入提赛德大学（University of Teesside）计算机科学系。在20世纪70~80年代，他编写了两本计算机体系结构领域的重要教材：《计算机硬件原理》（The Principles of Computer Hardware）和《微处理器系统设计》（Microprocessor Systems Design）。

2001年，他担任了计算机学会国际学生竞赛（CSIDC）主席，并于同年获得英国国家教学奖（National Teaching Fellowship），这是英国高等教育的最高奖项。由于在计算机体系结构教育方面的贡献，他于2002年获得IEEE CS本科教学奖，2006年获得IEEE CS 泰勒布斯教育奖（Taylor L. Booth award）。2009年被选为IEEE Fellow。他在IEEE计算机学会担任了多个职务，并积极参加课程体系设计，撰写了关于未来计算机体系结构教育的论文，参加了CS/ACM 2001计算课程体系的编写和制定工作。2010年Alan Clements从全职教学岗位退休。

CENGAGE Learning 圣智学习

CENGAGE Learning
www.cengageasia.com

投稿热线：(010) 88379604
客服热线：(010) 88378991 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn

封面设计 · 锡 彬

上架指导：计算机/计算机存储与外设

ISBN 978-7-111-55748-7

9 787111 557487 >

定价：79.00元

[General Information]
书名=计算机存储与外设
作者=（英）艾伦·克莱门茨（Alan Clements）
丛书名=计算机科学丛书
页数=240
SS号=14206532
出版日期=2017.03
出版社=北京：机械工业出版社
ISBN号=978-7-111-55748-7
中图法分类号=TP333
原书定价=79.00
主题词=电子计算机-外部设备-电子计算机-存储技术
参考文献格式=（英）艾伦·克莱门茨（Alan Clements）.计算机存储与外设[M].北京：机械工业出版社,2017.03.